Introduction to GFA-BASIC 32

After GFA-BASIC for MSDOS and GFA-BASIC for Windows 3.1 a 32-bits version of GFA-BASIC is available. These versions of GFA-BASIC were divided in a an interpreter and a separate compiler. Starting with GFA-BASIC 32 the interpreter part is replaced by an in-memory compiler, the same that is used to compile to external EXE files. When a program is run from the IDE the code is first compiled to machine code and then executed. The compiler is optimized for producing (very) fast machine code, so that GFA-BASIC 32 programs execute at high speed. The command library of GFA-BASIC 32 is partly compatible to GFA-BASIC 16-bit. Much of the functionality of the 16-bit version is retained, but due to an entirely new concept of creating and handling of windows and dialog boxes, GFA-BASIC 32 is also quite different and much more compatible to VB in that area. Other incompatibilities are due to the 32 bits operating system; an integer is now 32-bits wide instead of 16-bits in GFA-BASIC 16-bit, for instance.

One project-file only

GFA-BASIC (32-bit) code files are single project files. This is typical of GFA-BASIC since the very first version in 1985 for the Atari ST. Code, forms (windows and dialog boxes), data, resource info are all contained in one file; the .g32 source code file. To create modular programs code can be compiled in to a library file (.lg32) and included into the project file.

Editor

When possible GFA-BASIC 32 will automatically convert 16-bit code to the new 32-bit syntax. An odd number of parentheses in a code line are auto-completed to match all required parenthesis.

The underscore character (_) can be used to split "logical" lines of source code, across physical lines in the source code file. The effect of using a line continuation character is for "visual" appearance only - the compiler itself treats lines split this way as only one contiguous line of code. The colon character (:) can be used to separate multiple statements on a single (logical) line of source code. Subs and functions can be folded, of course. A rudimentary "intellisense: is provided for OCX objects.

Procedural

GFA-BASIC 32 is a procedural language and looks much like plain C, but it is syntax compatible with Visual Basic. GFA-BASIC 32 supports both the VB compatible Sub and FunctionVar statements, but it also provides the classical Procedure/Function statements from earlier BASICs and C.

Procedures and functions can have optional parameters. You can use GoSub and **Return** anywhere in a procedure, but GoSub and the corresponding Return statement must be in the same procedure.

New features

GFA-BASIC 32 has been greatly extended. Many new commands and functions are added, like ReDim, Iif, Choose, etc. New operators are included as well, like the conditional operator **?:**. It uses a question mark after the condition to be tested, and specifies two alternatives, one to be used if the condition is met and one if it is not. The alternatives are separated by a colon.

New data types

New data types are Large (64-bit), Date, Currency, Variant, Object, Pointer and Handle. Integer and Long data types are now 32-bits. New is the full support of 64-bit integer arithmetic. Variables declared without specifying a type explicitly are a **Variant** data type by default.

Array, Hash and Collection

Arrays can be redimmed now. Array elements can be inserted and deleted. The array can be sorted using quick sort in every possible way. The Hash is a one dimensional array or linked list whose (optional) index is of type string. The Hash list can be of any type, Int/String/Date/etc. A Hash is dynamic and is not dimensioned prior to its use. Values are added or assigned to existing elements. A hash can be examined, sorted, saved and loaded. Elements can be accessed by numeric index as well. Access to hash elements is very fast. The Hash is used with Split, Join, Eval and the regular expression functions reMatch, reSub.

The Collection is an COM object (OCX). It is a kind of one dimensional variant array whose index is of type variant. A collection is dynamic and is not dimensioned prior to its use. Values are added or assigned to existing elements. The collection is mainly targeted at OLE objects.

Const and Enum

A constant is a variable whose value is fixed at compile-time, and cannot change during program execution (hence, it remains constant). A constant is defined using the Const keyword. The Enum keyword is used to define a sequence of constants whose values are incremented by one.

Strings

For string functions the $-postfix is no longer mandatory, as in Chr$(0) which becomes Chr(0). The return value from a $-free string functions is NOT a Variant as in VB, but a real (ANSI) string. New are the Pascal compatible character constants #*number* that can be used in place of Chr(number). The following "Line1" #13#10 "Line2" #13#10 is the same as "Line1" + Chr(13) + Chr(10) + "Line2" + Chr(13, 10).

Besides + two new string concatenation operators are included: $ and &.

Comparison and assignment operators

In contrast with 16 Bit GFA-BASIC the expression x **==** y is now the same as x = y and x := y. The comparison operator == from16 Bit GFA-BASIC should now be replaced by NEAR. Alternatively, you can use a forced floating point comparison like If a = 1!.

Direct memory access

For direct memory access a whole range of variants of Peek and Poke are available (**PeekCur**, **PokeStr**, etc, etc.).

Memory move and manipulation commands are provided (MemMove, MemOr, MemAnd, MemBFill, etc).

’Bits and bytes’ swap and make functions (BSwap8, MakeHiLo, etc, etc).

Bits rotate and shift is supported (Shl, Shl8, Rol, etc).

Port access is supported (**Port Out**, **Port In**).

Matrix Arithmetic

Next to the normal arithmetic functions, GFA-BASIC 32 offers Matrix functions and many more (advanced) mathematical functions.
For runtime expression evaluation GFA-BASIC 32 includes Eval().

File functions

Special file functions are for checksums (Crc32, Crc16, CheckSum, CheckXor, etc), file encryptions (Crypt), file compression (Pack/UnPack). Others are MimeEncode/MimeDecode, MemToMime/MimeToMem, and UUToMem/MemToUU to convert between binary and plain text formats.

Built-in Win32 API functions

GFA-BASIC 32 supports more than 1000 API-Functions, functions that can be used as any other GFA-BASIC 32 function. Only the Standard-API-Functions from User, Kernel and GDI are implemented, other not often used API-Functions like for instance WinSock-Functions are to be declared explicitly.

The type of the parameters of the built-in API-Functions are not checked upon compiling. Each parameter is assumed to be a 32-bit integer. A string can be passed to an API function, but is always copied to one of the 32 internal 1030-Byte buffer BEFORE the address of the buffer is passed. A user defined Type (As type) is always passed by reference, so that its address is passed (automatically V:). To be on the safe side, keep things in your own hand and pass the addresses explicitly using VarPtr or **V:**.

These rules don't apply to DLL functions introduced with the Declare statement. Here GFA-BASIC 32 behaves like VB and the rules for calling such APIs must be respected. Some API function names are already in use by GFA-BASIC 32 and are therefore renamed. **GetObject**() becomes ***GetGdiObject***(), **LoadCursor** becomes ***LoadResCursor***. Obsolete functions are not implemented, obviously.

Built-in Win32 API constants

As with the built-in API functions from User, Kernel and GDI, their accompanying constants are built-in. (1400 API-Constants from the 16 Bit-Version and more then 900 Constants from Win32-APIs are implemented. Obsolete constants are not implemented, obviously.

Assembler and DisAssembler

GFA-BASIC 32 provides an inline assembler and a disassembler object (DisAsm).

Graphics

Graphic commands take floating point values (Single) now. After scaling (set with ScaleMode) the coordinates are passed as integers to the GDI system. Scaling provides much more flexibility and is VB compliant.

Most graphic commands can be used in VB format as well: Line (x, y)-(z, t),, BF is identical to Pbox x, y, z, t.

The Color-command is now the same as **RGBColor** in 16 Bit GFA-BASIC. Additionally, a table with the 16 standard colors can be used: **Color QBColor**(i) or a shortcut QBColor i.

The windows now have an AutoRedraw property so that output is captured (performance decrease) to a second bitmap as well. A redraw of the window is then performed by copying the contents of the bitmap to the screen.

Windows and dialogs

Windows and dialogs are all OLE - Forms now and their events are handled the same way as in VB. All standard and common controls are implemented using an OCX object wrapper. In general, all GUI objects are now OCX objects and are manipulated through properties, methods, and events. The old GetEvent/Menu() structure is now obsolete. You can still use third party controls by using the general **Control** statement. The notification messages are then handled in the window procedure of the parent, which is an form event sub as well!

COM programming

With CreateObject you create and return a reference to an ActiveX object. After the object reference is assigned to a variable of type Object you can use the object's properties, methods, and events.

Picture and Font Objects

OLE Object types to create and manipulate fonts and pictures. Since these types are OLE-type compatible of a Font or Picture instance can be assigned to a property of an OCX control or form. Other objects are:

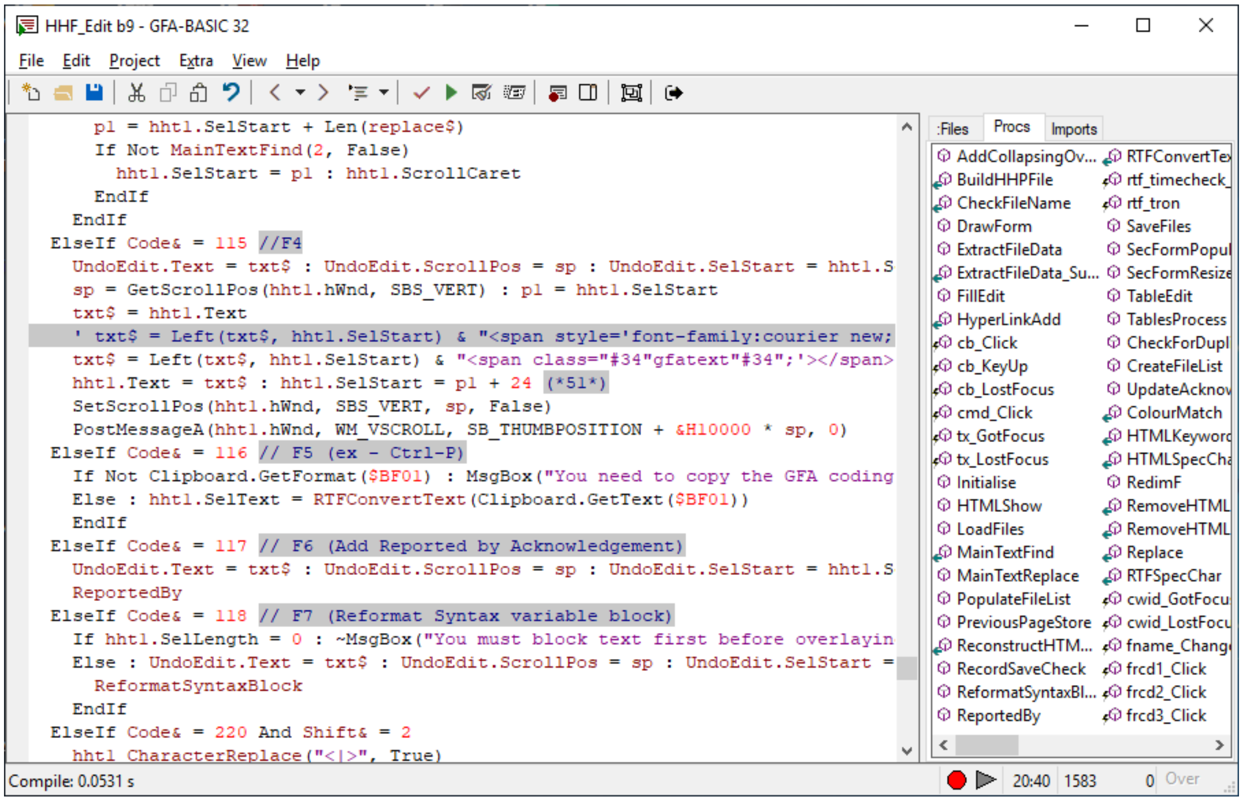
App Object	The App specifies information about the application's title, version information, the path and name of its executable file and Help files, and whether or not a previous instance of the application is running. In addition it provides methods to create shorcuts. It has many properties returning information that are otherwise hard to find.
Screen	Returns the current capabilities of the display screen your code is executing on. The Screen object has much more properties than the VB counterpart.
Err	Contains information about runtime errors or helps in generating useful errors. Try/Catch/EndCatch error-handling.
CommDlg	An OCX wrapper about the common dialog box functions.
Printer	Object that provides full printer support for your application.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

An Overview of the Integrated Environment

Starting GFA-BASIC from MS-Windows produces a standard Windows window which is used for editor input.

When you start GFA-BASIC 32, you see the interface of the integrated development environment, as shown below



```
      pl = hhtl.SelStart + Len(replace$)
      If Not MainTextFind(2, False)
        hhtl.SelStart = pl : hhtl.ScrollCaret
      EndIf
    EndIf
  ElseIf Code& = 115 //F4
    UndoEdit.Text = txt$ : UndoEdit.ScrollPos = sp : UndoEdit.SelStart = hhtl.S
    sp = GetScrollPos(hhtl.hWnd, SBS_VERT) : pl = hhtl.SelStart
    txt$ = hhtl.Text
    ' txt$ = Left(txt$, hhtl.SelStart) & "<span style='font-family:courier new;
    txt$ = Left(txt$, hhtl.SelStart) & "<span class="#34"gfatext"#34";'></span>
    hhtl.Text = txt$ : hhtl.SelStart = pl + 24 (*51*)
    SetScrollPos(hhtl.hWnd, SBS_VERT, sp, False)
    PostMessageA(hhtl.hWnd, WM_VSCROLL, SB_THUMBPOSITION + &H10000 * sp, 0)
  ElseIf Code& = 116 // F5 (ex - Ctrl-P)
    If Not Clipboard.GetFormat($BF01) : MsgBox("You need to copy the GFA coding
    Else : hhtl.SelText = RTFConvertText(Clipboard.GetText($BF01))
    EndIf
  ElseIf Code& = 117 // F6 (Add Reported by Acknowledgement)
    UndoEdit.Text = txt$ : UndoEdit.ScrollPos = sp : UndoEdit.SelStart = hhtl.S
    ReportedBy
  ElseIf Code& = 118 // F7 (Reformat Syntax variable block)
    If hhtl.SelLength = 0 : ~MsgBox("You must block text first before overlayin
    Else : UndoEdit.Text = txt$ : UndoEdit.ScrollPos = sp : UndoEdit.SelStart =
      ReformatSyntaxBlock
    EndIf
  ElseIf Code& = 220 And Shift& = 2
    hhtl CharacterReplace("<|>", True)
```

The GFA-BASIC IDE consists of two parts; the code editor and the Form Editor. When you first start GFA-BASIC 32 the code editor is visible. You can easily switch between the code and form editor using a toolbar button or the keyboard shortcut Shift + F7.

Note The figure shows the GFA-BASIC 32 IDE after it has been extended using the free available editor extension.

Menu Bar

Displays the commands you use to work with GFA-BASIC 32. Besides the standard File, Edit, View, and Help menus, a Project menu is provided to perform project specific tasks along with an Extra menu which can be used extend the environment with extensions or plug-ins, debugging and other tasks. More ...

Toolbar

Provides quick access to commonly used commands in the programming environment. You click a button on the toolbar once to carry out the action represented by that button. The toolbar can be customized by double clicking on an empty part of the toolbar.

Status bar

The bottom of the GFA-BASIC editor window contains a status bar line with various information. The larger left part is used for all kinds of information depending on the action currently taken. For instance, it shows the error messages which occur when the syntax control discovers an error.

Next to macro control panel, the time is shown. Then, besides the time panel, you'll find the cursor position (line : column). And finally, at right side you'll find the overwrite/insert mode indicator. The mode is toggled using the Insert key.

Code Editor

The GFA-BASIC editor is a program editor written especially for the development of GFA-BASIC programs. It is a line oriented editor, in that it performs a syntax check for each line, and it automatically indents the loops and subroutines. The syntax check means that the editor checks whether the entered statements are syntactically correct for GFA-BASIC. If not, a warning bell is sounded and the "Syntax error" message box is displayed. This can be switched off in the Properties window, though. In principle, a program line can have any length, but in practice only 7999 pixels will be displayed.

A comment can be placed in between statements or at the end of a GFA-BASIC statement. More ...

Sidebar

The Sidebar allows management of the program's inline data resources, procedures and imported files and, in the Form Editor, the properties and events associated with embedded OCX objects.

To activate the Sidebar either use Alt+4 or the toolbar button 'Split window'. The sidebar appears on the right and initially displays three tabs: ':Files', 'Procs', and 'Imports'; when the environment is switched to the Form Editor an additional 'Properties' tab window is created.

The Form Editor

The Form editor is a multiple document interface (MDI) that serves as a place to create forms. Here you design the interface of your application. You add controls and pictures to a form to create the look you want. Each form in your application has its own form designer window. More ...

Toolbox

The toolbox in only visible in the form editor mode. It provides a set of OCX tools that you use at design time to place controls on a form. The toolbox lists all Windows standard and common controls. More ...

{Created by Sjouke Hamstra; Last updated: 25/02/2019 by James Gaite}

The Menu Bar

The menu bar of the IDE is divided in several sub menus. Depending on the version of the IDE you are using, the organization of the menu items might differ - the description below is for version 2.54 onwards.

The File menu Show

Edit Menu Show

Project menu Show

Extra menu Show

View menu Show

Help menu Show

{Created by Sjouke Hamstra; Last updated: 26/02/2019 by James Gaite}

The Code Editor

The development environment includes an integrated Text editor to manage, edit, and print source files. Most of the procedures for using the editor will seem familiar if you have used other Windows-based text editors. In addition, GFA-BASIC 32 has enhanced the Text editor with several new timesaving features such as statement completion, dynamic syntax checking, and "intellisense" for OCX objects and event subs.

You can change many of the default settings for the Text Editor to conform to your preferences.

Folding

A particular feature of the GFA-BASIC editor is the ability to fold whole subroutines. The contents of these subroutines are then shown in the editor only by displaying the title line of the Procedure or Function. To indicate that this is a folded Procedure or Function the title line is prefixed with a greater-than character ">". To fold a Procedure or Function move the cursor into the subroutine (or its title line if folded) and enter:

F11 folds a Procedure, Sub or Function which has the cursor. Pressing F11 again unfolds the corresponding Procedure, Sub, or Function.

F12 folds all Procedures, Subs and Functions. Pressing F12 again unfolds all Procedures, Subs, and Functions.

Recording keys

Next to the information panel in the status bar you'll find a red dot and a grey arrow button. They provide the macro or key recording facility. It is able to store commands and characters. It stores most WM_COMMAND and WM_CHAR messages. The internal buffer for saving IDs and characters is 996 bytes long.

The status of the macro recording is reflected in the status bar with two buttons. The red circle indicates that currently nothing is being recorded. The arrow next to the circle is either solid black or gray. When black a macro is available and can be played back by clicking the arrow or by pressing Alt + Ctrl + P.

To start recording either click on the red circle or press Alt + Ctrl + R. Both Buttons will change, the red circle is replaced by a black rectangle representing the stop button, and the playback arrow is replaced by a pause button. Choosing pause skips recording until pause is selected again or when stop is selected. Selecting the stop button stops recording and redraws the red circle and arrow buttons. The arrow is now black filled: the macro can be replayed.

GLL Extension keyboard shortcuts.

GFA-BASIC 32 editor extension functions are often assigned to keyboard shortcuts to make them easily available. GFA-BASIC 32 has reserved 136 shortcuts to be assigned to a custom GLL extension event. These keyboard events are programmed by creating event subroutines with names that identify the keyboard shortcuts they must respond to. These keyboard subs have the fixed names Gfa_Ex_?, Gfa_App_? or Gfa_App_S?, where ? is a placeholder for one of the characters A-Z and the numbers 0-9. Thus, when you want to create an extension procedure that is invoked after pressing the combination Shift+Ctrl+X, the subroutine should be named Gfa_Ex_X.

Next:Keyboard Accelerators

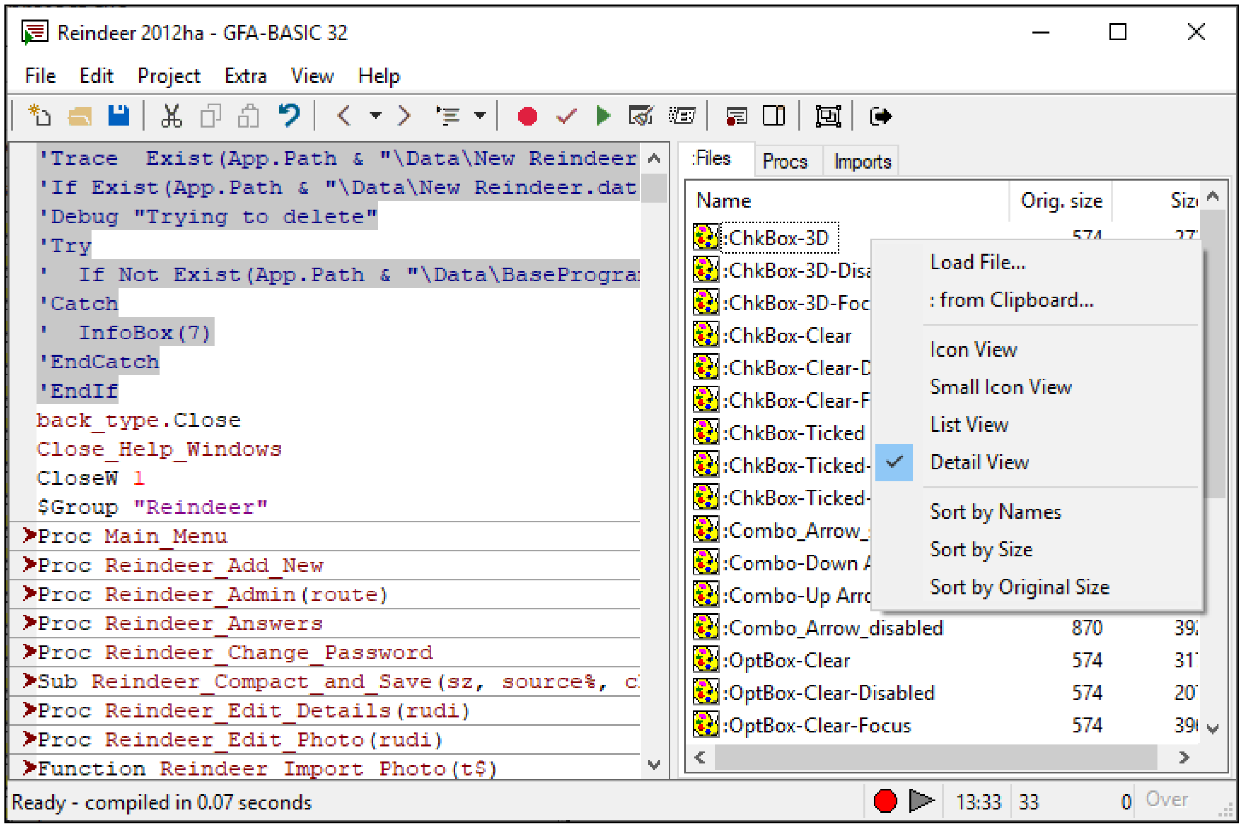
{Created by Sjouke Hamstra; Last updated: 27/02/2019 by James Gaite}

The :Files tab

General Description

GFA-BASIC 32 does not allow linking to Windows Resources, due to the fact that the 'interpreter-part' of the IDE performs in-memory compiling. For this reason, although the same compiler is used for external EXE files and can link resources, the in-memory compiler used in the Interpreter can not.

As an alternative GFA-BASIC 32 supports Inline resources, like it's predecessors have done since the early days of the Atari ST, and the ':Files' window allows you to manage them. You can add any kind of data file to this window and, later on, open it in your application. The data is packed using the GFA-BASIC 32 function Pack(), MimeEncode-d and then stored as ASCII characters within the source code file.



```
'Trace  Exist(App.Path & "\Data\New Reindeer
'If Exist(App.Path & "\Data\New Reindeer.dat
'Debug "Trying to delete"
'Try
'  If Not Exist(App.Path & "\Data\BaseProgra
'Catch
'  InfoBox(7)
'EndCatch
'EndIf
back_type.Close
Close_Help_Windows
CloseW 1
$Group "Reindeer"
Proc Main_Menu
Proc Reindeer_Add_New
Proc Reindeer_Admin(route)
Proc Reindeer_Answers
Proc Reindeer_Change_Password
Sub Reindeer_Compact_and_Save(sz, source%, c
Proc Reindeer_Edit_Details(rudi)
Proc Reindeer_Edit_Photo(rudi)
Function Reindeer_Import_Photo(t$)
```

Name	Orig. size	Siz
:ChkBox-3D	574	27
:ChkBox-3D-Disa		
:ChkBox-3D-Foc		
:ChkBox-Clear		
:ChkBox-Clear-D		
:ChkBox-Clear-F		
:ChkBox-Ticked		
:ChkBox-Ticked-		
:ChkBox-Ticked-		
:Combo_Arrow_		
:Combo-Down A		
:Combo-Up Arro		
:Combo_Arrow_disabled	870	39
:OptBox-Clear	574	31
:OptBox-Clear-Disabled	574	20
:OptBox-Clear-Focus	574	39

Ready - compiled in 0.07 seconds

Accessing & Viewing Inline Files

In GFA-BASIC 32 the Inline resources are accessed through I/O commands like **Open** #, **Input** #, etc. To differentiate the inline data from outside files, the names of the resources must start with a colon ':'. This way the GFA-BASIC 32 I/O commands recognize an inline file from an external file (a filename never starts with a colon) - for example, to load the graphic stored as 'ChkBox-3D' in the picture above, you would use the following code:

```
Dim graphic As Picture
graphic = LoadPicture(":ChkBox-3D") // Note the :
  before the filename
```

The ':Files' tab contains a ListView control and supports all user-interface options of such a control. You can move from one item to another using arrow, Page, Home, and End keys.

The information in the ListView control can be displayed in the following ways:

- Icon View - this dispalys the files as large icons with the internal names underneath;
- Small Icon View - the data files are listed using small icons with the internal name to the right - the entries are arranged from left to right, top to bottom;
- List View - similar to Small Icon View with the small icons and internal name to the right, but in this option the entries are arranged one below the other; and
- Detail View - this displays the internal name with the original and compressed size of the data file to the right of a small icon in a vertical list.

Switching between different view types is achieved by simply clicking with the right button in an empty region of the control and selecting the desired option from the right context menu; the same menu can be used to sort the chosen view by Name, Compressed Size, or Original Size.

Adding & Previewing Data

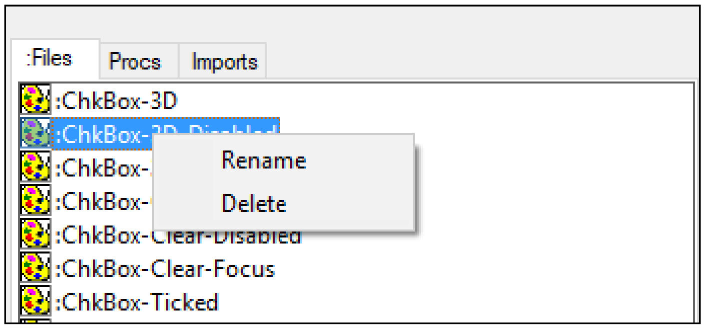
To add data, you can either drag and drop a file on to the Files window, use the option "Load file" (accessed through the right click menu), which opens a file-select dialog box to select a file, or ": From Clipboard", which copies the contents of the clipboard to the ":Files" tab. When you choose to add data from the clipboard, a context menu pops up containing a summary of all objects currently available on the system clipboard. Select the format of the data you wish to add and a dialog box pops up to specify a name for the ':Files' resource. Although you could remove the colon at the beginning of the string, it will be inserted when the name is accepted and the data is added to the resource section.

If you hover your mouse over an entry, the bitmap, icon, cursor, or enhanced meta file, is made visible after a second.

Deleting & Renaming Data

You can delete a resource object by selecting its name and either press Delete, or right-click on it to get a context menu. Select Delete from the menu.

To rename an object, right click its name and select Rename.



Known Issues

1. *[Fixed in version 2.54]*It has been reported that, very rarely, if 'Detail View' is selected from the right click menu, no file information is displayed. It has not been possible to reproduce this bug or find out what causes it. [Reported by James Gaite, 20/02/2019]
2. If the sort order is changed and then 'Small Icon View' is selected, the spacing in between the files is not even. This can be rectified by simply selecting a different viewing option and then returning to 'Small Icon View'. [Reported by James Gaite, 20/02/2019]
3. *[Fixed in version 2.54]* The original documentation mentions the ability to drag and drop files into the Files Window. At present, although the mouse cursor changes while carrying out such an operation, this is not possible and it is not known whether this was ever implemented. [Reported by James Gaite, 23/02/2019]

See Also

The Procs tab, The Imports tab, The Properties Tab

{Created by Sjouke Hamstra; Last updated: 26/02/2019 by James Gaite}

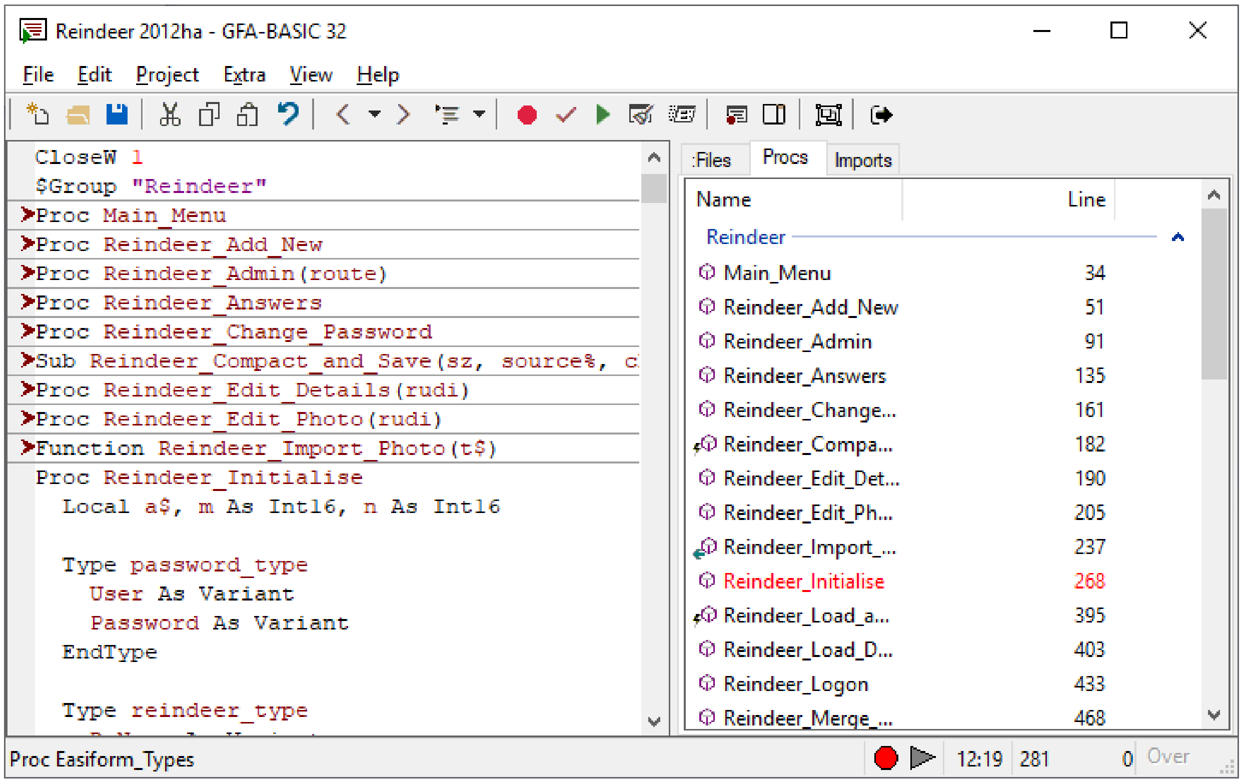
The Procs tab

General Description

The 'Procs' window shows and gives quick access to all procedures and functions of your application. For more information on Procedures, see here.

As can be seen from the figure below, all sub routines are represented by a cube, with functions being distinguished by a blue arrow pointing to the left, signifying the fact that they return a value, and subs by a small flash of lightning, signifying that they are used for events.

The routine containing the caret is highlighted in red; left-clicking on another one moves the caret to the first line of that routine. In addition, as with all ListView objects, once you have selected a routine it is possible to navigate using the arrow keys, Home and End, and select using 'Enter'.


Reindeer 2012ha - GFA-BASIC 32
File Edit Project Extra View Help
Files Procs Imports
Proc Easiform_Types
12:19 281 0 Over

```
CloseW 1
$Group "Reindeer"
Proc Main_Menu
Proc Reindeer_Add_New
Proc Reindeer_Admin(route)
Proc Reindeer_Answers
Proc Reindeer_Change_Password
Sub Reindeer_Compact_and_Save(sz, source%, c
Proc Reindeer_Edit_Details(rudi)
Proc Reindeer_Edit_Photo(rudi)
Function Reindeer_Import_Photo(t$)
Proc Reindeer_Initialise
  Local a$, m As Int16, n As Int16

  Type password_type
    User As Variant
    Password As Variant
  EndType

  Type reindeer_type
```

Name	Line
Reindeer	
Main_Menu	34
Reindeer_Add_New	51
Reindeer_Admin	91
Reindeer_Answers	135
Reindeer_Change...	161
Reindeer_Compa...	182
Reindeer_Edit_Det...	190
Reindeer_Edit_Ph...	205
Reindeer_Import_...	237
Reindeer_Initialise	268
Reindeer_Load_a...	395
Reindeer_Load_D...	403
Reindeer_Logon	433
Reindeer_Merge_...	468

Grouping

Introduced in Version 2.5, it is now possible to group routines in user-defined blocks: this greatly facilitates the navigation of the Procs list, especially in large programs.

Creating a group is done by placing the statement **$Group *Groupname*** immediately above the first line of the first procedure you wish to include in the block. An example of this can be seen in the picture above where the `$Group "Reindeer"` statement immediately precedes the *Main_Menu* procedure creating the group which you can see in the Procs window to the right. This block will contain all subsequent procedures until either another $Group statement or the end of the program listing is reached.

Groups can be collapsed and expanded using the small arrow to the right, as well as by double-clicking on the group name. Also, once the caret enters a routine in a gropuping, that group is automatically unfolded in the Procs window and, if the 'Collapse Inactive' option is selected in the Right-click menu, the other group is automatically folded. By default, all groups except that containing the caret are collapsed.

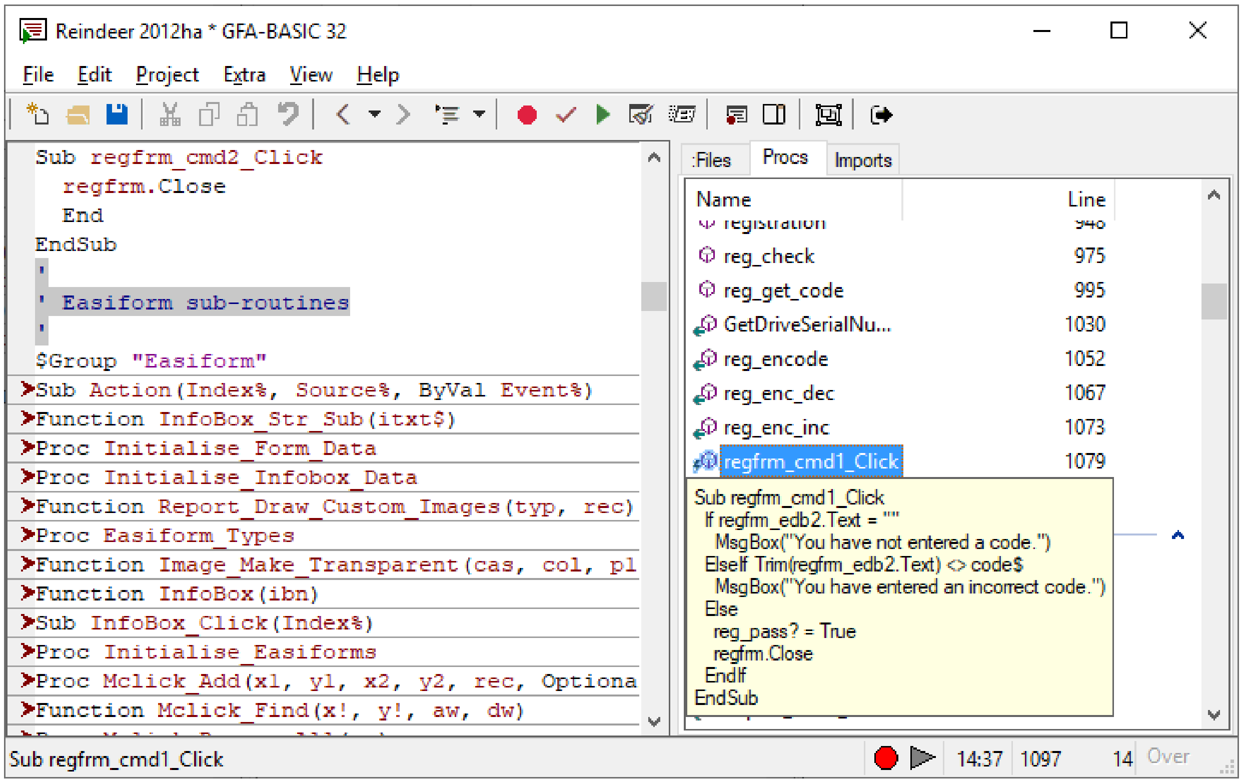
Due to the way GFABASIC handles the $Group command, simply deleting the line containing the $Group statement will not remove that group; instead, to permamently delete the block, you must replace the whole $Group statement - including the group name - with the **$GroupOff** command, while it is possible to temporarily disable a group by inserting a blank line between the $Group command and the first line of the first routine. To temporarily disable all groups, deselect 'GroupView' in the right-click menu.

IMPORTANT:

1. Grouping only works in 'Detail View'.
2. When grouping is used, it is necessary to add a $Group statement immediately preceding the first routine; if this is not done, all routines above the first $Group statement are omitted from the list.
3. If a group is not created, check that the relevant $Group statement is on the line immediately above the first procedure you wish to include in the group.
4. If no groups are shown in the Procs window, ensure that both 'Detail View' and 'GroupView' are selected in the right-click menu.

PeekView

Another feature added in Version 2.5 is the ability to 'Peek' at the code of a certain procedure by using the Procs window. This is done by hovering the mouse pointer over the routine's name and using the mousewheel to expand or contract the listing, similar to the method used to view coding in a folded procedure in the main edit window. Below is example of PeekView in use:


Reindeer 2012ha * GFA-BASIC 32
File Edit Project Extra View Help
Files Procs Imports
Name Line

```
Sub regfrm_cmd2_Click
  regfrm.Close
  End
EndSub
'
' Easiform sub-routines
'
$Group "Easiform"
Sub Action(Index%, Source%, ByVal Event%)
Function InfoBox_Str_Sub(itxt$)
Proc Initialise_Form_Data
Proc Initialise_Infobox_Data
Function Report_Draw_Custom_Images(typ, rec)
Proc Easiform_Types
Function Image_Make_Transparent(cas, col, pl
Function InfoBox(ibn)
Sub InfoBox_Click(Index%)
Proc Initialise_Easiforms
Proc Mclick_Add(x1, y1, x2, y2, rec, Optiona
Function Mclick_Find(x!, y!, aw, dw)
```

Name	Line
registration	948
reg_check	975
reg_get_code	995
GetDriveSerialNu...	1030
reg_encode	1052
reg_enc_dec	1067
reg_enc_inc	1073
regfrm_cmd1_Click	1079

```
Sub regfrm_cmd1_Click
  If regfrm_edb2.Text = ""
    MsgBox("You have not entered a code.")
  ElseIf Trim(regfrm_edb2.Text) <> code$
    MsgBox("You have entered an incorrect code.")
  Else
    reg_pass? = True
    regfrm.Close
  EndIf
EndSub
```

Sub regfrm_cmd1_Click   14:37 1097 14 Over

Further Options

List View or Detail View - Accessed through the right-click menu, there are two options available as to how to display the routines in the Procs window. List View sorts them into columns of names as tall as the open window while Detail View displays them in one list of two columns, the first containg the routine's name and the second the line number of its first line. **NOTE:** Grouping does not work in List View.

Goto Proc/Select Proc - Once again, these options are accessed through the right-click menu. Selecting either will unfold the selected routine and display it in the Edit Window - the latter option will then select or block the whole procedure. This Goto Proc action can also be achieved by double-clicking on the routine's name in the Procs window.

Print Proc - Also accessed through the right-click menu, this option performs the same task as **Select Proc** and then sends the code listing for that procedure to the printer.

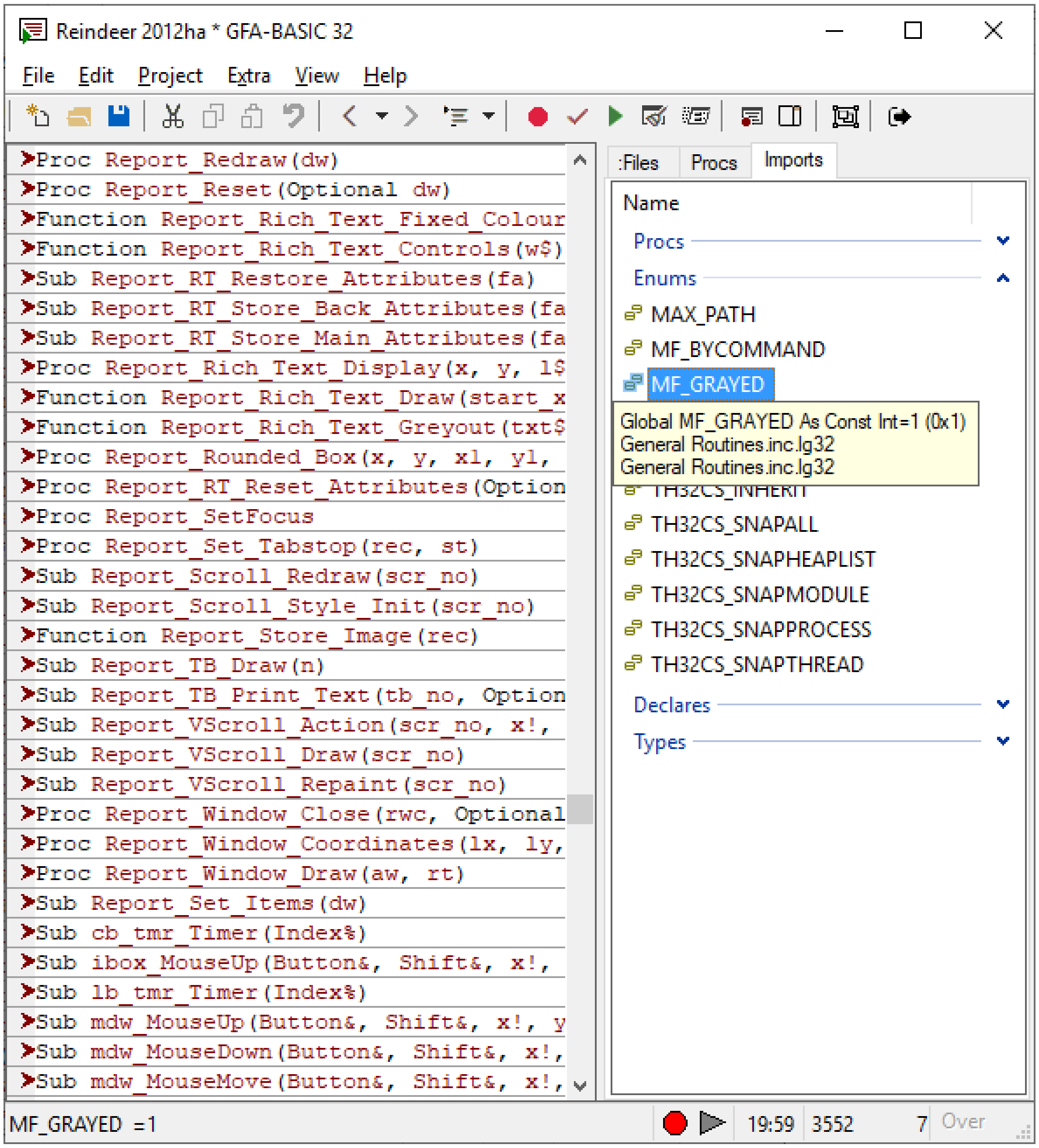
See Also

The Files tab, The Imports tab, The Properties Tab

{Created by Sjouke Hamstra; Last updated: 02/03/2019 by James Gaite}

The Imports tab

The 'Imports' windows displays all elements that are imported using the $Library command for compiled GFA-BASIC 32 library files (.lg32).



Reindeer 2012ha * GFA-BASIC 32

```
Proc Report_Redraw(dw)
Proc Report_Reset(Optional dw)
Function Report_Rich_Text_Fixed_Colour
Function Report_Rich_Text_Controls(w$)
Sub Report_RT_Restore_Attributes(fa)
Sub Report_RT_Store_Back_Attributes(fa
Sub Report_RT_Store_Main_Attributes(fa
Proc Report_Rich_Text_Display(x, y, l$
Function Report_Rich_Text_Draw(start_x
Function Report_Rich_Text_Greyout(txt$
Proc Report_Rounded_Box(x, y, x1, y1,
Proc Report_RT_Reset_Attributes(Option
Proc Report_SetFocus
Proc Report_Set_Tabstop(rec, st)
Sub Report_Scroll_Redraw(scr_no)
Sub Report_Scroll_Style_Init(scr_no)
Function Report_Store_Image(rec)
Sub Report_TB_Draw(n)
Sub Report_TB_Print_Text(tb_no, Option
Sub Report_VScroll_Action(scr_no, x!,
Sub Report_VScroll_Draw(scr_no)
Sub Report_VScroll_Repaint(scr_no)
Proc Report_Window_Close(rwc, Optional
Proc Report_Window_Coordinates(lx, ly,
Proc Report_Window_Draw(aw, rt)
Sub Report_Set_Items(dw)
Sub cb_tmr_Timer(Index%)
Sub ibox_MouseUp(Button&, Shift&, x!,
Sub lb_tmr_Timer(Index%)
Sub mdw_MouseUp(Button&, Shift&, x!, y
Sub mdw_MouseDown(Button&, Shift&, x!,
Sub mdw_MouseMove(Button&, Shift&, x!,
```

Name
Procs
Enums
MAX_PATH
MF_BYCOMMAND
MF_GRAYED

```
Global MF_GRAYED As Const Int=1 (0x1)
General Routines.inc.lg32
General Routines.inc.lg32
```

TH32CS_INHERIT
TH32CS_SNAPALL
TH32CS_SNAPHEAPLIST
TH32CS_SNAPMODULE
TH32CS_SNAPPROCESS
TH32CS_SNAPTHREAD
Declares
Types

```
MF_GRAYED =1          19:59  3552   7  Over
```

General Description

As can be seen from the picture above, imported elements are displayed in collapsible or foldable groups according to their type which are:

- Procs (including Functions);
- Variables (including Hash Arrays);
- Enums (and Constants);
- Declares (APIs); and
- Types

Initially the 'Procs' group, showing all the imported Procedures and Functions, is the only 'unfolded' group.

More Information

More information can be gleaned from the Imports list by both hovering over and clicking the relevant entry.

Clicking on the entry displays the name of the entry along with any optinal description in the IDE's status; in addition, when an 'Enum' entry is clicked, the value of the entry is also displayed.

Hovering the mouse over an entry gives even more information in a small pop-up box next to the item (see the picture above). The information differs according to the item type as follows:

- **Procs** - The Procedure, Function or Sub declaration, the description (if added in the Library) and finally the name of the parent library.
- **Variables and Enums** - The Varaible declaration including Type and initial value, the description (if added in the Library) and then the name of the parent library.

- **Declares** - The API declaration, the description (if there is one) and the parent library./LI>
- **Types** - The full Type declaration showing element names and types, the optional description (if any) and, lastly, the parent library.

Known Issues

There are a few problems when displaying details of an item when the mouse hovers over it:

1. *[Fixed in version 2.54]* With all import types (except Types), if there is no optional description, the library name is also omitted.
2. *[Fixed in version 2.54]* For a variable or a constant, the library name is given twice, once after the variable declaration line AND then again at the end.
3. *[Fixed in version 2.54]* For a Declare, the optional description appears twice after the declaration line.
   [Reported by James Gaite, 20/02/2019]

See Also

The Files tab, The Procs tab, The Properties Tab

{Created by Sjouke Hamstra; Last updated: 26/02/2019 by James Gaite}

The IDE Properties

The properties window is real system Properties Dialog box and thus the window text is displayed in your language (here Dutch).

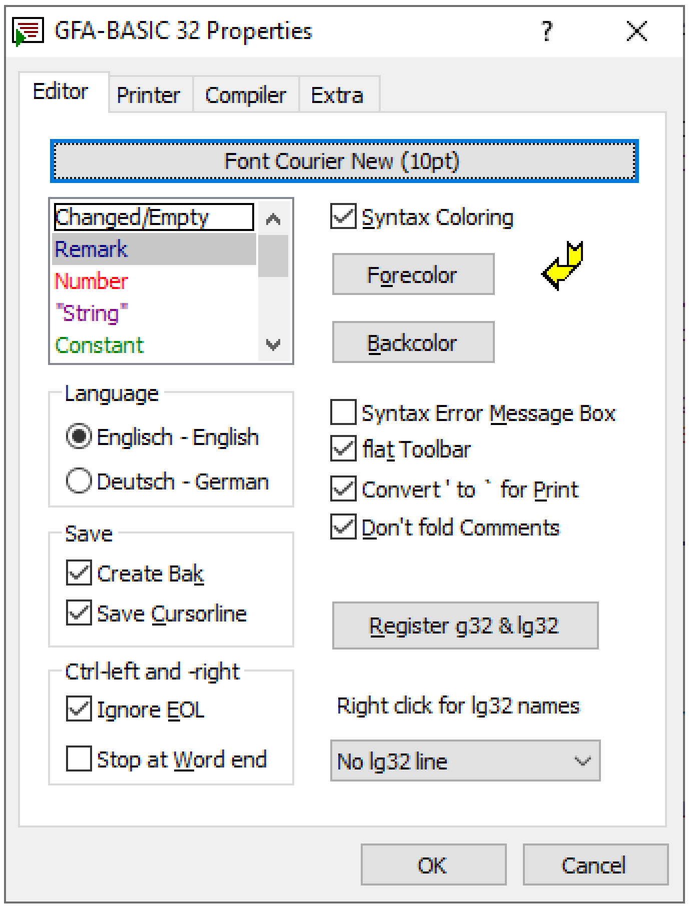
The dialog box displays three or more tabs, depending on the version you own. The Editor tab allows you to set code editor options, the Printer tab provides settings for printing a code listing, and the Compiler tab is used to set compiler options.

The Editor Tab

Syntax Formatting

The top elements in the Editor Tab all affect the display of the syntax in the Code Editor.

- The long button at the top decribes the current font attributes of the text - clicking this opens up a Font Select window allowing you to change the appearance to suit your taste.
- **Syntax Coloring** - If left unticked, the syntax in the Code



GFA-BASIC 32 Properties

Editor | Printer | Compiler | Extra

Font Courier New (10pt)

Changed/Empty
Remark
Number
"String"
Constant

[✓] Syntax Coloring
Forecolor
Backcolor

Language
(●) Englisch - English
( ) Deutsch - German

Save
[✓] Create Bak
[✓] Save Cursorline

Ctrl-left and -right
[✓] Ignore EOL
[ ] Stop at Word end

[ ] Syntax Error Message Box
[✓] flat Toolbar
[✓] Convert ' to ` for Print
[✓] Don't fold Comments

Register g32 & lg32

Right click for lg32 names

No lg32 line

OK | Cancel

Editor is displayed in black; if ticked, then elements of the syntax are differentiated from one another by their font colour. The listbox to the left of this check box contains all the categories of syntax and shows their current respective colours. These can be edited by selecting one of the elements in the listbox then clicking on either the **Forecolor** or **Backcolor** buttons to the right, depending upon which aspect you wish to change; a pop-up box of different colours is then displayed from which a new colour can be selected.

IDE Language

Below the Syntax Coloring you are given the option to change the **Language** in which elements of the IDE and messages are displayed. Currently the only two options are English and German - English is the default UNLESS German is the default UI.

Save Options

In the **Save** frame are two options to alter IDE behaviour when a file is saved:

- **Create Bak** - When the option is selected the old version of the file isn't deleted when you save a program file. The existing file will be renamed (the extension. "bak") before the file is saved. An already existing .bak file will be deleted first.
- **Save Cursorline** – When selected the cursor position is stored in the file when it is saved.

Ctrl-Left and -Right

Using Ctrl with the left and right directional arrows allows quick movement through a line a code. The default behaviour is for the cursor to be moved in the direction of the arrow used, skipping to the beginning of the next/preceding word until the end of the line is reached when you can move no further. Included in this section of the Editor Properties are two options to alter this behaviour as follows:

- **Ignore EOL** - When selected the cursor doesn't stop at the end of a line.
- **Stop at word end** - When selected the cursor also stops at the end of a word not the beginning of the next (or preceding).

Miscellaneous Options

Along the right of the box below the Syntax Coloring buttons are six miscellaneous options which affect the Editor in the following ways:

- **Syntax error Message Box** - When selected a Syntax Error Message Box is displayed when a code line contains a syntax error. The code must be repaired before the line can be left. The cursor is placed at the character that causes the error. When not selected a line can be left even when it contains syntax errors. The code line is then displayed using the error foreground color. Afterwards a line with a syntax error can be located by pressing (Shift+) F4.
- **Flat Toolbar** - Makes the toolbar flat - in previous version sof Windows, the toolbar buttons would appear raised if this option was de-selected; however, the only difference made in Windows 10 is that the dividing lines between the different button groups disappear.

- **Convert ' to ` for Print** - When selected in a code statement starting with the command **Print** the ' (comment) character is converted to a ` (a space character). When not selected the ' character is interpreted as a comment.
- **Don't fold Comments** - This option prevents that comment lines at the end of a procedure are folded with the subroutine. Lines, with ' or / at the start of a line, immediately before the start of the next subroutine, are then not folded. They remain visible between the folded procedures. These comment lines are then used to optically separate subroutines.
- **Register g32 & lg32** - Registers the document types .g32 and .lg32 for the currently running GFA-BASIC 32 IDE instance. Icons for g32 and lg32 file types are registered that are displayed in front of GFA-BASIC 32 files.
  Windows provides file associations so that an application can register the type of documents it supports. The benefit of doing this is that it allows the user to double-click or select a document in the Explorer to edit it. Registering the file associations is one step procedure performed by the user; GFA-BASIC 32 doesn't register the document types itself.
- **Right click for Lg32 names** - Shows the name of the LG32 filename when an imported name is clicked with the right Mouse button (in the editor). The name of the lg32 file can be displayed above or below the imported name.

See Also

Printer Tab, Compiler Tab, The Extra Tab

{Created by Sjouke Hamstra; Last updated: 26/02/2019 by James Gaite}

The Form Editor

To create a window or dialog box, you must create a form to contain controls, add controls to the form, set properties for the controls, and write code that responds to form and control events.

·To activate the Form Editor either use F7 or click the button on the toolbar.

A new, empty Form is displayed. Use the Properties window to set properties for the Form - that is, to change the name, behavior, and appearance of the form. For example, to change the caption on a form, set the Caption property.

Add a control

Use the Toolbox to add controls to the form. The Toolbox is always visible in the Form Editor. To see the name of a particular control in the Toolbox, position the mouse pointer over that control.

To add a control, find the control you want to add in the Toolbox, drag the control onto the form, and then drag one or more of the control's adjustment handles until the control is the size and shape you want. The element is placed with a default size. However, to size a control while you add it, place the pointer where you want the upper-left corner of the control to be, then drag to the right and down until the control is the size you want.

To copy a control hold down the Ctrl key and click the control to be copied.

To delete a control or form simply select the control that you wish to delete and then press the Delete-key. You can also right-click on the control or form and select Kill Ocx/Form from the context menu. A Message box will appear to make sure whether you really want to delete the item. Answering 'Yes' will delete the item permanently.

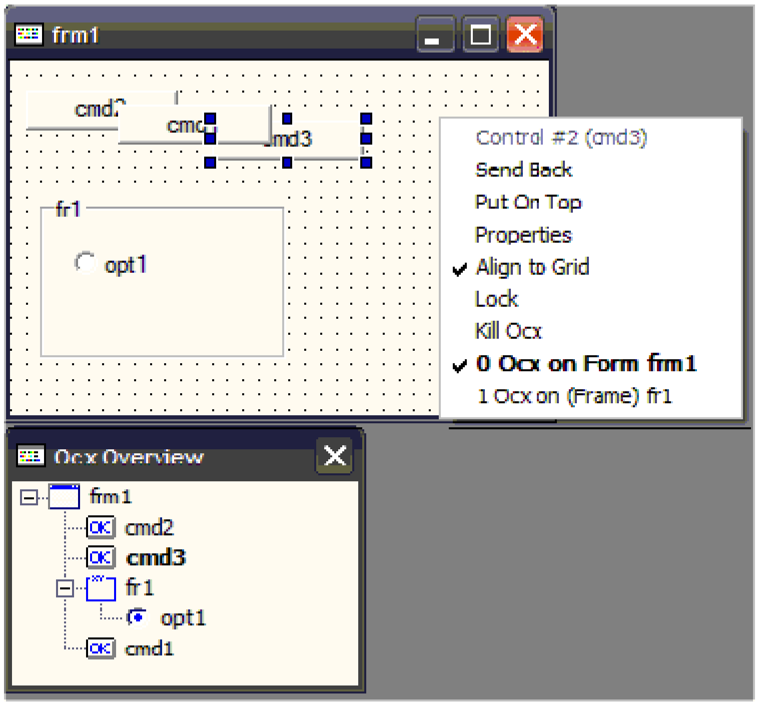
After you've added controls to the form, use the commands in the context menu (right click) to adjust the alignment and spacing of the controls. Hold down the Shift-key to select multiple controls.

The context menu

The Form editor contains special tools for layout. These tools help align and arrange controls in the correct place. The Form editor tools are collected in a context menu, which differs from situation to situation. The context menu shows the tools applicable for the selected form, control, or controls.

The first line of the context menu displays the form or control(s) that are affected. This line is dimmed so that it cannot be selected.

One of the options from the context menu is "Align to Grid". When you are placing or arranging controls in a form, you can use the layout grid (8 pixels) for more precise positioning. When this option is turned on, controls appear to “snap to” the dotted lines of the grid as if magnetized. You can turn this “snap to grid” feature on and off and change the size of the layout grid cells.



Tab order

Use the 'Ocx Overview' dialog box (View menu) to set the tab order of the controls on the form. GFA-BASIC 32 determines the tab order by the order the controls are placed on the form (there is no *TabIndex* property). The tab order can be changed by dragging the controls in the 'Ocx Overview' dialog box. If you want to prevent users from tabbing to a particular control, you can set the **TabStop** property to **False** for that control, but only in code.

Order of creation

The tab order determines order of creation. The last control created has the topmost attribute, it is placed highest in the Z-order and is drawn over other controls. The visibility or the Z-order can be set using the context menu of the selected control. A partly visible control can be put on top by selecting "Put on Top", and a control at the top of the Z-order can be placed behind another control by selecting

"Send Back". The tab order is immediately updated in the "Ocx Overview" window. The control number (#) in the context menu shows the creation order.

Ocx on Ocx

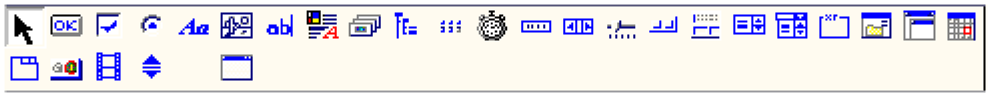
Some OCXs can be used as parent control. Assigning controls to a parent makes sure that they are in the correct Z-order and that they can be moved together with their parent. To assign a control to a parent, select the parent BEFORE adding a control to the form. Right-click somewhere in the form and select from the context menu the parent of the next control. In the figure above, the new control will be added to Form frm1 directly. To add the control to the Frame fr1 select "Ocx on (Frame) fr1". The Ocx Overview window reflects the owner-child relationship by adding a branch to the parent OCX. See also OcxOcx

Next:The Toolbox
Creating a Control
Setting OCX Properties

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

The Toolbox

Use the Toolbox to add controls to the form. The Toolbox is always visible in the Form Editor. To see the name of a particular control in the Toolbox, position the mouse pointer over that control.



The first icon in the Toolbox displays a mouse pointer. The pointer is meant for resizing controls and changing their attributes. To change the size of an element, simply click on one of the corners of an element and resize the surrounding box.

The toolbox contains the following OCX controls:

Ocx Control	**Description**
Command	A pushbutton is the best known button available in Windows. It is just a button where the user can click on. As long as the mouse-key is not released, the button's visible state is changed (it looks 'pressed' down).
Option	Also known as Radio button. Option buttons are normally 'grouped' together in a Groupbox (Frame). Only one radio button can be selected within one group.
Checkbox	normally can be switched either off (empty) or on (with an X-mark in it). Use this control to give the user a True/False or Yes/No option.
Label	displays text that a user can't change

	directly.
Image	control can display a graphic from a bitmap, icon, or metafile, as well as enhanced metafile, JPEG, or GIF files.
Textbox	displays information entered at design time, entered by the user, or assigned to the control in code
RichEdit	enter and edit text while also providing more advanced formatting features than the conventional TextBox control.
ImageList	a repository of images for use by other OCXs and by controls with a Picture property.
TreeView	displays data in a hierarchical in nature
ListView	displays data as ListItem objects. Each ListItem object can have an optional icon associated with the label of the object.
Timer	execute code at regular intervals by causing a Timer event to occur.
ProgressBar	graphically represents the progress of a transaction.
Scroll	easy navigation through a long list of items or a large amount of information.
Slider	consists of a scale, defined by the Min and Max properties, and a "thumb," which the end user can manipulate using the mouse or arrow keys.
ToolBar	a frame containing a collection of Button objects.
StatusBar	a frame that can consist of several panels which inform the user of the status of an application.
ListBox	displays a list of items from which the user

	can select one or more.
ComboBox	combines the features of a TextBox control and a ListBox control-users can enter information in the text box portion or select an item from the list box portion of the control.
Frame	provides an identifiable grouping for controls. You can also use a Frame to subdivide a form functionally-for example, to separate groups of Option controls.
CommDlg	provides a standard set of dialog boxes for operations such as opening and saving files, setting print options, and selecting colors and fonts. The control also has the ability to display help by running the Windows Help engine
Form(control)	a control with the same features as a Form (use as PictureBox for instance).
MonthView	to view and set date information via a calendar-like interface.
TabStrip	acts like the dividers in a notebook or the labels on a group of file folders. By using a TabStrip control, you can define multiple pages for the same area of a window or dialog box in your application.
TrayIcon	creates a taskbar notification icon.
Animation	displays silent Audio Video Interleaved (AVI) clips.
UpDown	a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a *buddy* control.
Form	New Form (or press Shift + F7 or select the menu item 'New Form')

Several OCX controls can be used as parent OCX:

Image - A container with a small resource footprint. This could be used instead of a Form, which uses more resources (scaling, a DC, a Picture).

Form - A Form OCX can be used as a container (of course).

Frame - Particular useful for Option OCXs (Radio Buttons). The **.Transparent** property of the Frame may not be changed; otherwise, the embedded controls are invalid.

TabStrip - To embed (for instance) a Frame Ocx.

ToolBar - To embed (for instance) a ComboBox Ocx.

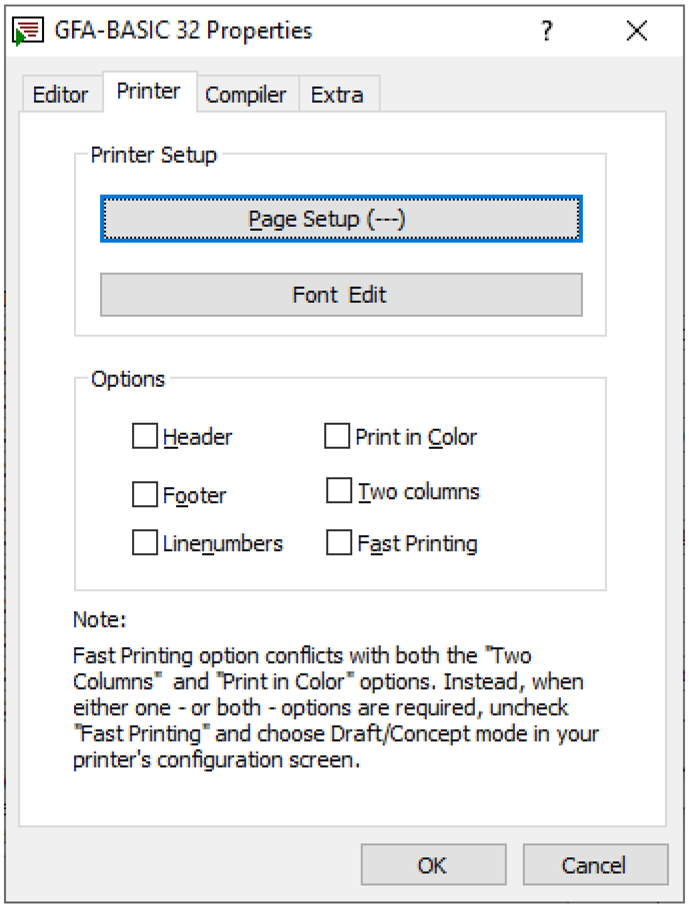
StatusBar - To embed (for instance) a Command Ocx

Next:Creating a Control

Setting OCX Properties

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Printer Properties



☐ Header ☐ Print in Color
☐ Footer ☐ Two columns
☐ Linenumbers ☐ Fast Printing

Note:

Fast Printing option conflicts with both the "Two Columns" and "Print in Color" options. Instead, when either one - or both - options are required, uncheck "Fast Printing" and choose Draft/Concept mode in your printer's configuration screen.

Page Setup

Allows the selection of paper size, source, page orientation and margins.

Font ...

Allows selection of a font for printing. If no font is selected, displays 'Font Edit'; otherwise is displays the selected font name.

Options

- **Header** - Enables a page header of the print job.
- **Footer** - Enables a page footer of the print job.
- **Linenumbers** - Enables the printing of line numbers.
- **Print in Colo**r - Printing in color mode is a bit slower, and usually requires more space to spool. On monochrome printers BASIC statements and function are printed bold, comments in italics, error lines bold-underlined, and declared DLL-Functions underlined.
- **Two columns** - Two column printing. Useful with (mainly) short lines of code (or possibly small fonts)
- **Fast Printing** - Enables fast printing. The result is, depending on the printer and driver, usually a little faster

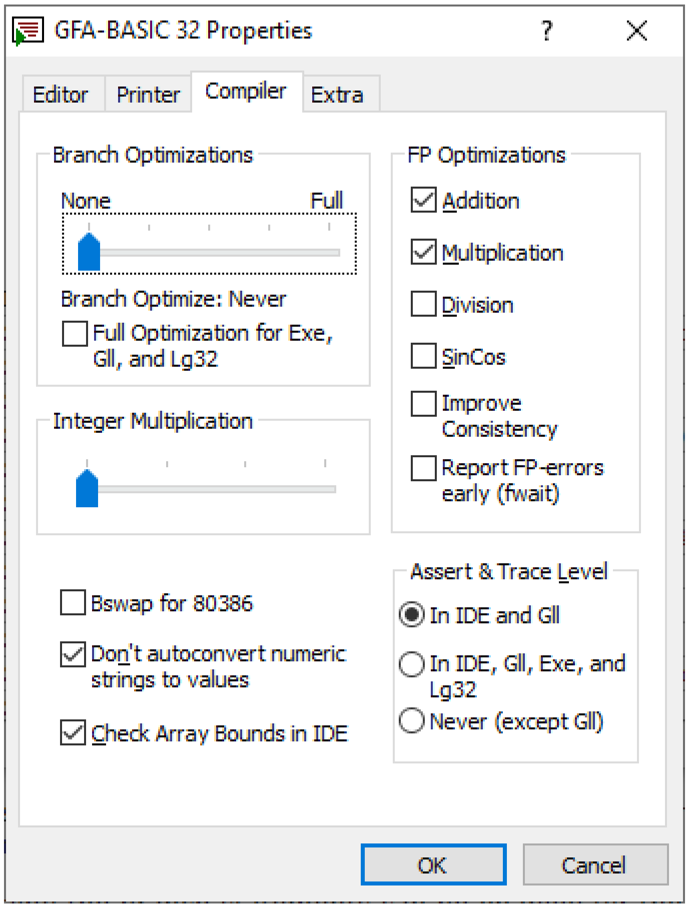
than usual, and it produces smaller spooling files. In this mode, the color and two-column modes are disabled. Page and footer options are allowed.

See Also

Editor Tab, Compiler Tab, Extra Tab

{Created by Sjouke Hamstra; Last updated: 25/02/2019 by James Gaite}

Compiler Properties



GFA-BASIC 32 Properties

Editor | Printer | Compiler | Extra

Branch Optimizations

None — Full

Branch Optimize: Never

- [ ] Full Optimization for Exe, Gll, and Lg32

Integer Multiplication

FP Optimizations

- [✓] Addition
- [✓] Multiplication
- [ ] Division
- [ ] SinCos
- [ ] Improve Consistency
- [ ] Report FP-errors early (fwait)

- [ ] Bswap for 80386
- [✓] Don't autoconvert numeric strings to values
- [✓] Check Array Bounds in IDE

Assert & Trace Level

- ● In IDE and Gll
- ○ In IDE, Gll, Exe, and Lg32
- ○ Never (except Gll)

OK | Cancel

Many of the properties on this page relate to optimizations relevant to much older processors than you will find in modern computers (386, 486 and early Pentiums).

Branch Optimizations

BranchOpt - The optimization of branch statements provides for a slightly smaller program size and increases the performance a little bit. However, compile time increases, and therefore this option can disabled partially or completely. Suggestion: Disable (None)

Full Optimization for Exe, GlI & Lg32 - The time consuming process of optimizing the branch statements can be enabled for the creation of stand-alone EXEs only. Compiling in memory in the IDE will not result in branch optimization. Suggestion: Disable

Integer Multiplication

Here you can adjust how much code the BASIC must generate, to the avoid the multiplication statement of the 80x86 processor when a value is multiplied by a constant. GFA-BASIC 32 replaces the code through a series of Shl, Add, Sub and Lea statements. This requires more code. However, modern processors are already optimized.

The slider setting 1 = left allows only very short replacement codes (10 bytes). This is the setting when you need a smaller program. A slightly higher performance is obtained with the second setting (up to 20 bytes increase) on a Pentium 100. The third setting (up to 30 bytes) is on a Pentium a little slower and the program a little larger, however for a 486 bit this setting results in faster execution. (The Pentium is, of course, still faster than the 486, but slower than the same Pentium with setting 2). The fourth setting inserts sequences to 40 bytes.

The settings: 1 = space, 2 = optimization for Pentium, 3 = optimization for 486, 4 = especially suitable for programs that are often used on 486's.

Here with Pentium all Pentiums are meant, except the old Pentium 60/66 Conventions, as well as the i486-DX4. These processors have rapid multiplication built-in.
This setting only affects programs that are using many integer multiplications with constants. Suggestion: Disable

Bswap for 80386

This option is for the rare case that a program uses the Bswap function and the program is run on a 80386 computer. Normally, the processor is able to execute the Bswap statement (the exchange of the four byte in an integer) very quickly. However, for these the Bswap is emulated using xchg al, ah : . rol eax, 16 : . xchg al, ah.

Not only is this emulation longer in code size, but also significantly slower. Note that 80386-computers running Windows NT or Windows 95 are not numerous. Suggestion: Disable

Don't autoconvert numeric strings to values

This option is for the not so rare case that a programmer wants some more control over Type conversions. In the past BASICs have always converted between different numeric formats automatically (Double <-> Int etc.). VBA introduced an automatic conversion of string values to numeric (an OLE internal Val) and vice versa. This automatic conversion can lead to difficulty in finding bugs, and is in conflict with prior versions of GFA-BASIC. With this optionselected the automatic conversion of strings in numbers is disabled. This does not apply to operations with Variants or Objects. Suggestion: Enable.

Check Array Bounds in IDE

Inserts code to check for every access to an array to determine if the index is within the range of the array. If the index is not within array bounds an error message is displayed. Un-selecting this option turns off the array bounds error checking and removes checks for the correct number of dimension of the array. **Note** This may speed up array manipulation but invalid memory locations may be accessed and result in unexpected behavior or program crashes.

FP Optimizations

Note Enabling these optimizations may prevent the correct execution of your program.

Addition - If this option is chosen, floating-point additions, or subtractions, are calculated at the compile level. For example:

a = a + 100 - 99 is compiled to a = a + 1.

This usually has no influence on the result of calculations. However, through the limited number of digits of the internal floating-point values, there will be a small divergence as the sum (a, 100 and -99) vary in magnitude. So if a is smaller than _epsDbl*100, so a+100==100. But if a is greater than _epsDbl, then a+1 is not equal to 1, however mathematically correct. Without this optimization only the first two digits of variable a are considered. Only some mathematical calculation methods will be affected. Almost all programs can have this optimization set. Suggestion: Enable

Multiplication - Like the optimization option for floating-point additions and subtractions, there is an optimization for multiplication. Negative effects for this optimization should be even less than that of the addition/subtraction, because the multiplication is more insensitive to magnitude differences. Suggestion: Enable

Division - This option optimizes floating-point division. Division by a constant is replaced by a multiplication of the reciprocal of the constant. For example:

a = a / 10 becomes a= a * 0.1

However, since the number of digits of a Double data type is limited, and the computer works with binary values, the value 0.1 cannot not accurately be represented, so that the division by 10 and multiplying it by 0.1 returns slightly different results (deviation around 1E-16). A reciprocal of division of values of the power of two results in an exact floating-point value (like /8 and *0125). The reason for this optimization is, as almost always, the speed. Suggestion: Disable

SinCos - The Intel processors (Pentium, 486, 387th ..) have built-in functions for sine, cosine and tangent. The values of these functions is to +- 2^63. Greater values (> 1E18) don't result in an error message from the processor, and the return value is 0. A program must test this explicitly. Using this setting uses the processor functions of Sin / Cos / Tan, which are minimally faster (and minimally shorter), but without this test. A value range overrun for Tan leads then to a floating-point stack error and in the case of Sin / Cos mostly to completely absurd errors, because the argument is returned unchanged (Sin (1E40) = 1E40?) Suggestion: Disable

Improve Float Consistency

The Improve Float Consistency option improves the consistency of floating-point tests for equality and inequality by disabling optimizations that could change the precision of floating-point calculations.

By default, the compiler uses the coprocessor’s 80-bit registers to hold the intermediate results of floating-point calculations. This increases program speed and decreases program size. However, because the calculation involves floating-point data types that are represented in memory by less than 80 bits, carrying the extra bits of precision (80 bits minus the number of bits in a smaller floating-point type) through a lengthy calculation can produce inconsistent results.

With this option the compiler loads data from memory prior to each floating-point operation and, if assignment occurs, writes the results back to memory upon completion. Loading the data prior to each operation guarantees that the data does not retain any significance greater than the capacity of its type.

When some other floating point instruction is executed before the IF clause, the equality test is correct, though. To force a reload you could use: ~0�much some instruction. Suggestion: Enable.

Report FP-errors early - This option results in a small increase of the size and program execution time. If checked, the compiler generates the fwait assembly statement, in particularly with a conversion of floating-point values to integer values. A floating-point error is not reported before the next floating-point statement or in a fwait. The impact of this option is that an (overflow) error is reported a little earlier. Only in extreme cases, the activation of this option is useful for finished programs. Suggestion: Disable

Assert and Trace Level

Determines whether Assert, Trace or Debug.Print code is inserted:

In IDE and Gll - Only when the program is compiled in memory in the IDE or as a Gll.

In IDE, Gll, Exe and Lg32 - Inserts code for Assert / Trace / Debug.Print in both the IDE and in the EXE/GLL/LG32 output file.

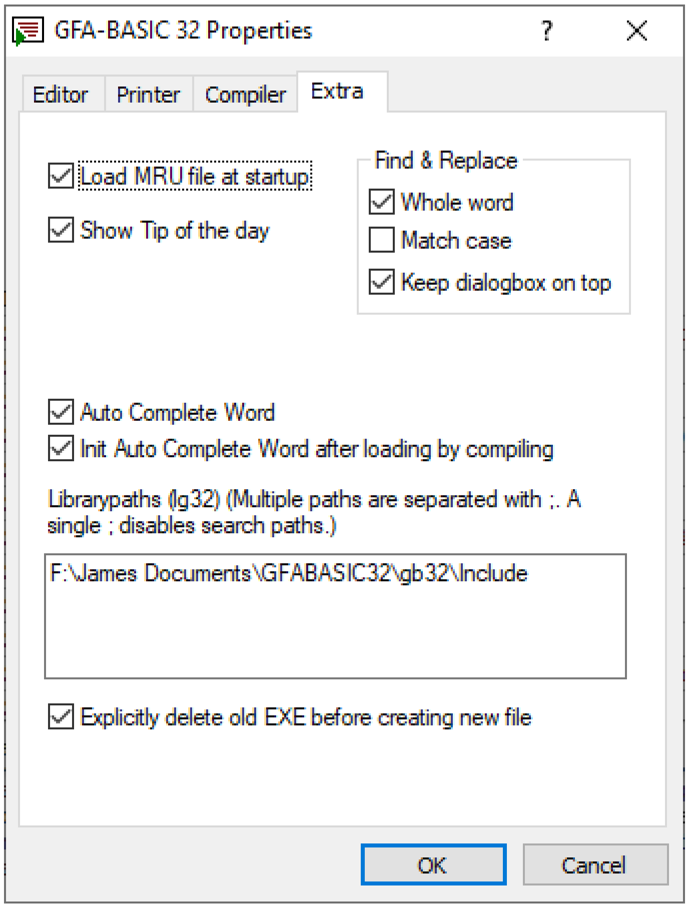
Never (except Gll) - No insertion of Assert / Trace / Debug.Print code at all except in Gll output files.

See Also:

The Editor Tab, The Printer Tab, The Extra Tab, Compile To Exe Tab

{Created by Sjouke Hamstra; Last updated: 25/02/2019 by James Gaite}

IDE Extra Properties



GFA-BASIC 32 Properties

[✓] Load MRU file at startup
[✓] Show Tip of the day

Find & Replace
[✓] Whole word
[ ] Match case
[✓] Keep dialogbox on top

[✓] Auto Complete Word
[✓] Init Auto Complete Word after loading by compiling

Librarypaths (lg32) (Multiple paths are separated with ;. A single ; disables search paths.)

F:\James Documents\GFABASIC32\gb32\Include

[✓] Explicitly delete old EXE before creating new file

The Extra Menu contains miscellaneous options which do not accurately fit into the description of the other three tabs. These are:

Load MRU file at startup

If selected, the last program you were working on will automatically be loaded at startup.

Show Tip of the Day

The first time the program is started each day, a tip is shown. To stop this, deselect this option.

Find & Replace

The first two options in the this frame set the default behaviour for Find & Replace - **Whole Word** when selected will only look for whole words that match the search syntax and **Match Case** when selected will execute a case-sensitive search - while the last option **Keep dialogbox on top** allows the search box to remain visible and on top of the Code window at all times.

Auto Complete Word

Auto Complete Word turns on the Auto-Complete function in the Code Editor. As there are certain variables which can not be added to the Auto-Complete list before a Compile is performed due to the way that the IDE was originally written - User-defined Types are one example - a second option **Init Auto Complete after loading by compiling** will force a compile immediately after the project is loaded into the IDE so that all values are then available. *Note* that, if there is a compile error during this procedure, any variables defined after that point will still not be avaialble.

Librarypaths (lg32)

By default this text box lists the GFABASIC Include folder; other folders containing user libraries can be added separated by a semi-colon (;).

Explicitly delete old EXE before creating new file

When a compiled EXE file is created, GFABASIC overwrites any existing EXE files of the same name by default. Very rarely, this will force a 'File Write' error and this option is included to get around this: if selected, instead of overwriting the file, GFABASIC deletes the old EXE first before creating the new one. There are no downsides to having this optin selected.

{Created by James Gaite; Last updated: 28/02/2019 by James Gaite}

Creating an application

The first step to creating an application is to create the interface, the visual part of the application with which the user will interact. Forms and controls are the basic building blocks used to create the interface; they are the OCX objects that you will work with to build your application.

Note - In GFA-BASIC 16 bit the interface was created using windows and dialog boxes that were created using special commands like **OpenW**, **ChildW**, **ParentW**, and **Dialog**. These commands are still available, but create **Form** objects as well. GFA-BASIC 32 application windows are always **Form** objects now.

OCX objects

Forms and controls are wrapped in OLE objects, called OCX objects. OLE controls are also known as OCX controls or ActiveX controls. However an OLE object doesn't need to have a visible aspect; it may be invisible at run time. OCX is a natural development of the older VBX extension that use older technology and are found in applications written in earlier versions of Visual Basic.

GFA-BASIC 32 implements all standard and custom controls, forms (windows and dialog boxes), and many other features in OCX objects. OCX objects are kind of object-oriented objects wrapped using OLE techniques. OLE controls are often provided in dynamic link libraries with an .OCX extension. That's why the run-time library of GFA-BASIC 32 is called GfaWin32.OCX.

Forms are OCX objects that expose properties which define their appearance, methods which define their behavior, and events which define their interaction with the user. By setting the properties of the form and writing code to respond to its events, you customize the object to meet the requirements of your application.

Controls are OCX objects that are contained within form objects. Each type of control has its own set of properties, methods, and events that make it suitable for a particular purpose. Some of the controls you can use in your applications are best suited for entering or displaying text. Other controls let you access other applications and process data as if the remote application was part of your code.

You work with forms and controls, set their properties, and write code for their events at *design time*, which is any time you're building an application in the GFA-BASIC 32 environment. *Run time* is any time you are actually running the application and interacting with the application as the user would.

Next:Using Forms

Using OCX Controls

Using Event Procedures

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Using Forms

Forms are the foundation for creating the interface of an application. You can use forms to add windows and dialog boxes to your application. You can also use them as containers for items that are not a visible part of the application's interface. For example, you might have a form in your application that serves as a container for graphics that you plan to display in other forms.

Properties

Many of a form's properties affect its physical appearance. The **Caption** property determines the text that is shown in the form's title bar; the **Icon** property sets the icon that is displayed when a form is minimized. The **MaxButton** and **MinButton** properties determine whether the form can be maximized or minimized. By changing the **BorderStyle** property, you can control the resizing behavior of the form.

Height and **Width** properties determine the initial size of a form; **Left** and **Top** properties determine the form's location in relation to the upper left-hand corner of the screen. The **StartupMode** property can be set to start the form in a maximized, centered, or normal state.

The **Name** property sets the name by which you will refer to the form in code. By default, when a form is first added to a project, its name is set to *frm1*, *frm2*, and so forth. It's a good idea to set the **Name** property to something more meaningful.

Many form properties correspond with other control properties that you can examine in Using OCX Controls. The form, however, is unique in that it does not reside on a form, but appears on the user's window. That is why the form's Left, Top, Width, and Height properties all correspond to the edge of the screen and not to a Form window.

In addition to the properties shared with the controls, the form has - among others -the following properties:

BorderStyle	This property determines how the Form window responds to the user's efforts to resize it. Some values you may need are 0-None, which offers a form without any edge or title bar, 1-Fixed Single, which offers a non-sizable window (the user can close the window but not resize, minimize, or maximize the window), and 2-Sizable (the default), which offers a regular sizable window with maximize and minimize buttons.
ControlBox	This property's value of True or False determines whether the form's Control menu appears. A Control menu is the menu that appears when you click a window's icon in the upper-left corner of the window. The Control menu enables you to move, size, minimize, maximize, and close a window.
Icon	This property specifies an icon filename for the Windows taskbar icon that appears when the user minimizes the form.
MaxButton	This property determines whether the form contains an active Maximize window button.
MinButton	This property determines whether the

	form contains an active Minimize window button. (If you set both the MaxButton and MinButton properties to False, neither appears on the form.)
Movable	This property determines if the user can move the form or if the form is to remain in its displayed location.
Sizeable	This property determines if the user can size the form.
ShowInTaskbar	This property's True or False value determines whether the open form appears on the user's Windows taskbar.
StartUpMode	This property provides a quick way to specify the starting position of the form on the screen. One of the most useful values is 2-Center that centers the form on the user's screen when the form first appears.

Load a form

To make a form visible and make your application run, you would use the following piece of code:

```
LoadForm frm1
Do
  Sleep
Until Me Is Nothing
```

This loads the form settings from the internal data and brings it on the screen. The Do loop makes sure that it stays active. When the form is closed the **Me** variable will no longer reference a valid form and the loop will end. **Me** always holds the current active form.

- Now, press F5 to run the program.

Forms can perform methods and respond to events.

The **Resize** event of a form is triggered whenever a form is resized, either by user interaction or through code. This allows you to perform actions such as moving or resizing controls on a form when its dimensions have changed.

The **Activate** event occurs whenever a form becomes the active form; the **Deactivate** event occurs when another form or application becomes active. These events are convenient for initializing or finalizing the form's behavior. For example, in the **Activate** event you might write code to highlight the text in a particular text box; in the **Deactivate** event you might save changes to a file or database.

Next:Using OCX Controls

Using Event Procedures

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Using OCX controls

Many of the controls require similar properties as forms. The next table lists some common properties that most controls support. All controls have a screen location (indicated by the Left and Top properties) and a size (indicated by the Width and Height properties), and most have foreground and background colors as well as font properties, if the controls display text.

Property	**Description**
Alignment	Determines whether text on the control, such as a label or command button, is left-justified, centered, or right-justified on the control.
BackColor	Specifies the color of the control's background, which you select from a palette of colors when you open the property's drop-down list box of colors.
BorderStyle	Determines whether the control has a border around it.
Caption	Lists the text displayed on the control.
Enabled	Set by a drop-down list box, this property is either True if you want the control to respond to the user or False if you want the control not to respond to the user. This property is useful for turning on and off controls when they are and are not available during a program's execution.
Font	Displays a Font dialog box from which you can set various font properties, such as size and style, for a control's text.
ForeColor	Specifies the color of the control's

	foreground, which you select from a palette of colors when you open the property's drop-down list box of colors.
Height	Specifies the number of twips high the control is.
Left	Indicates the starting twip from the left edge of the form where the control appears. For a form, the Left property specifies the number of twips from the left edge of the screen.
MousePointer	Determines the shape of the mouse cursor when the user moves the mouse over the control at runtime.
Name	Specifies the name of the control. As you saw in yesterday's lesson, the Properties window displays the Name property in parentheses so that it appears first in the list of properties.
ToolTipText	Holds the text that appears when the user rests the mouse cursor over the control at runtime.
Top	Is the starting twip from the top edge of the form where the control appears. For a form, the Top property describes the number of twips from the top edge of the screen.
Visible	Set by a drop-down list box, this property is True if you want the control to be visible on the form or False if you want the control to be hidden from view.
Width	Specifies the number of twips wide that the control is.

Some control properties, such as the Alignment property values, may look strange because their drop-down list boxes display numbers to the left of their values. For example, the Alignment property can take on one of these three values: 0 'Left Justify, 1 'Right Justify, and 2 'Center. You can use your mouse to select these values from the list without worrying about the numbers in them, but you can also, after opening the drop-down list box for a property, type the number that corresponds to the value you want to quickly set that value. The numbers also come in handy when you assign property values to controls with code.

Control Focus

Only one control on a form can have the focus at any one time. The first control with the focus is determined by the order in which you placed the controls on the form or, more accurately, the order determined by the creation order of each control on your form. This can be modified using the "Ocx Overview" window.

Not every control can receive focus. Only those controls the user can interact with can receive the focus. For example, a label control cannot receive the focus because the user cannot interact with label controls. The focus, or control focus, is the currently selected control. The control with the focus is indicated by highlighting the control.

Next:Using Event Procedures

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Using Event Procedures

Event procedures sometimes challenge beginning GFA-BASIC 32 programmers, but the concept of an event procedure is very simple. When the user presses a command button or enters text into a text box, something has to happen that tells the application the user has just made a move. Windows receives events from all kinds of sources. Most events come directly from the user at the keyboard and mouse running applications within Windows.

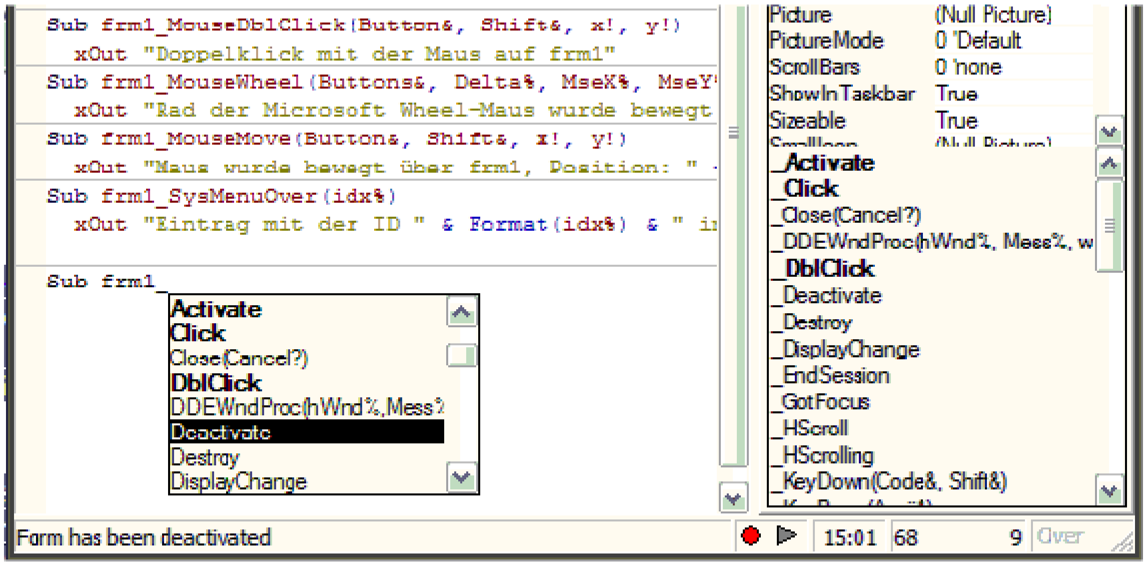
When Windows recognizes that the user triggered an event and that the event is not a system event, such as a Windows Start button click, but an event directly needed by an application, Windows passes that the event to the application. If you've written an event procedure to respond to that exact event, your application will respond to the event. If you haven't written an event procedure, the event goes unhandled.

Creating Event Procedures

Code in a GFA-BASIC 32 application is divided into smaller blocks called *procedures*. An *event procedure* contains code that is executed when an event occurs (such as when a user clicks a button). An event procedure for a control combines the control's actual name (specified in the **Name** property), an underscore (_), and the event name. For example, if you want a form named *frm1* to invoke an event procedure when it is clicked, use the procedure Sub frm1_Click.

One way to create an event procedure, is to select the name of a form in the Properties sidebar. The second half of the sidebar window displays all event subs for the form.

Select the name of an event for the form. Note that a *template* for the event procedure is now displayed in the Code window.



```
Sub frm1_MouseDblClick(Button&, Shift&, x!, y!)
  xOut "Doppelklick mit der Maus auf frm1"
Sub frm1_MouseWheel(Buttons&, Delta%, MseX%, MseY
  xOut "Rad der Microsoft Wheel-Maus wurde bewegt
Sub frm1_MouseMove(Button&, Shift&, x!, y!)
  xOut "Maus wurde bewegt über frm1, Position: "
Sub frm1_SysMenuOver(idx%)
  xOut "Eintrag mit der ID " & Format(idx%) & " i

Sub frm1_
```

Activate
Click
Close(Cancel?)
DblClick
DDEWndProc(hWnd%,Mess
Deactivate
Destroy
DisplayChange

Picture	(Null Picture)
PictureMode	0 'Default
ScrollBars	0 'none
ShowInTaskbar	True
Sizeable	True

_Activate
_Click
_Close(Cancel?)
_DDEWndProc(hWnd%, Mess%, w
_DblClick
_Deactivate
_Destroy
_DisplayChange
_EndSession
_GotFocus
_HScroll
_HScrolling
_KeyDown(Code&, Shift&)

Form has been deactivated

15:01 68 9 Over

Another way to create an event procedure is by typing the Sub statement at the beginning of a line, and then type the OCX name followed by an underscore. A list box with all event names pops up and lists all available and already used events (bold). A short description is displayed in the status bar.

The underscore separates the OCX name from the event name and is required. All event procedures are named this way. Therefore, an event procedure named cmdExit_DblClick () executes if and only if the command button named cmdExit's event named DblClick occurs.

Common OCX Events

You should familiarize yourself with common events that can occur for the controls that you know about. Both the form and its controls can receive events. Here are some common form events that can occur during an application's execution:

Activate	This event occurs when a form gets the focus. If an application contains multiple forms, the Activate event occurs when the user changes to a different form by clicking on the form or by selecting the form from a menu.
Click	This event occurs when the user clicks anywhere on the form. If the user clicks a form that's partially hidden from view because another form has the focus, both a Click and an Activate event take place.
DblClick	This event occurs when the user double-clicks the form.
Deactivate	This event occurs when another form gets the focus. Therefore, both the Activate and Deactivate events occur when the user selects a different form. You may choose to write event procedures for both events for each form, for only one event for one of the forms, or a combination thereof depending on the needs of your application.
Load	This event occurs right as the form is loaded into active memory (using **LoadForm**!)and appears on the screen.
Paint	This event occurs when Windows must redraw the form because the user uncovered part of the form from under another object, such as an icon.
Resize	This event occurs when the user changes the size of the form.
Destroy	This event occurs when the application

removes a form from the window using code. When an application is terminated, all loaded forms are first unloaded, so you must write an Unload event procedure for each form if you want to perform some kind of clean-up or file-saving procedure at the end of an application's session.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Debugging Described

Since the early days GFA-BASIC implements its own debugging facilities by using a trace line concept (**Tron**). This concept has proven its use, because it provides a quick and direct way to inspect a piece of code or variables by including a part of code in a **Tron**/**Troff** block.

When the **Tron** *proc* statement is included in a program, the procedure *proc* is executed before each source code line that is to be executed next. The redirection to *proc*, which must be part of your program, starts as soon as the **Tron** command is executed.

To facilitate the **Tron** functionality the 'call tronproc' assembler instruction is compiled into a program. The assembler instruction is inserted before each code line, but only when the code isn't between the **$StepOff** and **$Step** commands. The tron call instruction is not compiled into external EXE files.

By default the tron call invokes an empty GFA-BASIC 32 library function that returns immediately, so does nothing. As long as the **Tron** command isn't used, the overhead is kept to a minimum this way. When the **Tron** *proc* is executed, the empty library function is replaced by the specified tron procedure and will be called before executing of the next line(s).

The **Tron** procedure has access to information about the running program through the following debugging functions:

TraceLnr returns the number of the line that is to be executed next.

Trace$ returns the text of the current program line.

SrcCode$(line%) returns the text of the specified line.

ProcLnr(pname$) returns the line number of the first line of the subroutine with the specified name.

ProcLineCnt (p$) returns the number of lines of the specified subroutine

EdShowLine Lnr% shows the Tron-arrow in the margin of the specified line and pauses 0.5 seconds.

adr%=TraceReg returns the address of the memory block containing the stored processor registers. The registers are saved on the stack just before invoking the Tron procedure and are restored after leaving the Tron proc. The order in which they are saved is:

edi esi esp ebp ebx edx ecx eax efl eip.

The value of ebp is obtained like this: Debug.Print LPeek(adr+4*4)

TraceReg (reg) returns the contents of the specified register from the TraceReg memory block. The argument *reg* can be one of the 32-bit registers eax, ebx, ecx, edx, ebp, esp, esi, edi, efl, and eip.
The 16 bit registers ax, ax, ax, ax, ap, ap, ai, ai, fl, as well as the 8 bit registers al, bl, cl, dl, ah, bh, ch, dh.
Note When writing (LPoke) to the registers you should not change esp and ebp.

With these commands a simple (or complex) debugger can be created. The main disadvantage is that you must merge the **Tron** procedure into your code each time you need it.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

The built-in IDE debugger

With the editor extension commands for debugging, you can manipulate the default debugger, which is started when a program is run (F5). The debugger is implemented as an invisible window that creates a tray icon and that responds to the messages from the tray icon. Simultaneously, a second thread is started to respond to the Ctrl-Break and Shift-Ctrl-Break keyboard shortcuts, for which a system wide keyboard hook is created. After the program is ended, the invisible debug window is destroyed and so is the tray icon. Unfortunately, the icon often remains visible, but this is simply a 'Windows thing' and cannot be blamed on GFA-BASIC 32. The thread responsible for the Ctrl-Break shortcuts remains, but has no purpose any longer: the system wide keyboard hook is removed. The next time a program is run all debug settings are initialized to the GFA-BASIC 32 default settings.

Using the tray icon

By default the debugger doesn't do very much. As soon as a project is executed, GFA-BASIC 32 creates a second thread to monitor a Ctrl-Break to stop the program and a tray icon to provide some basic debugging facilities. It provides the means to step through the code, either one line a time or auto step (follow) where the line is marked a very short time (100 ms). It allows to continue or to pause your program. The tray icon menu is opened by right clicking the tray icon. The menu also let you open the debug Output window.

Starting to step through code is only possible by left clicking the tray icon or selecting Step in the tray icon menu.

Unfortunately, the ability to click comes at a time the program has already initialized and has come to its main message loop. To start debugging from the beginning of a program, a **Stop** or **MsgBox** is required to hold the program and give you some time to activate the debugger. Once the debugger is started, it allows stepping through the code by clicking with the left mouse button on the tray icon. Alternatively, the program flow can be 'followed', which is the same as normal running, but with the debug arrow visible.

Using the tray icon programmatically

An editor extension can invoke the debugger tray icon commands as well. **Gfa_DbStep** enters stepping mode so that you can step through the code. If you want to step through the code starting from the beginning, put **Gfa_DbStep** in the **Gfa_OnRun** event sub. If you would like to start stepping on a 'breakpoint', use it in a **Gfa_TronBook** event sub. GFA-BASIC 32 provides commands for the other tray icon functions as well. To pause an application you would use **Gfa_DbOn** and to continue use **Gfa_DbOff**. When using these commands you can still use the default meaning of the left mouse button, namely stepping through the code.

Besides the commands to switch the debugger mode, GFA-BASIC 32 provides the facility to intercept the tray icon mouse clicks. To install a click event sub you would use the **Gfa_DebOn** method.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

The good old Tron

Another way to implement debugging in a GFA-BASIC program is by using the Tron statement. Tron *proc* has been available since the GFA-BASIC for MSDOS version. When the **Tron** *proc* statement is included in a program, the procedure *proc* is executed before each source code line that is to be executed next. The redirection to *proc* starts as soon as the **Tron** command is executed. As a programmer you insert the **Tron** statement at the point the trace must start. This command has proven its use, because it provides a quick and direct way to inspect a piece of code or variables.

The **Tron** facility is by default available to any program that is run from within the IDE. GFA-BASIC 32 compiles the code with calls to a Tron procedure. If **Tron** isn't used the call returns immediately. The insertion of a **Tron** proc call takes a few extra bytes before each code line. To disable the insertion of Tron ready code, you must use **$StepOff**.

A GLL Tron

A GLL **Tron** *gll_proc* runs in the context of the IDE (different thread and main message loop) and must manipulated in a different manner. To start redirecting the code to the tracing *gll_proc* you cannot put the **Tron** statement in your code, because *gll_proc* is not visible to the code in the program. To make use of a general GLL you must go a different way.

To install a GLL Tron procedure you must use **Gfa_Tron** *proc,* which installs a procedure (located inside the same

GLL) to be executed before each statement. There can only be one **Gfa_Tron** registered with the IDE.

The question now is how to start the trace and start invoking call *proc_gll*?

The **Gfa_Tron** proc statement is only useful after starting (Run, F5) the program. One of the first steps GFA-BASIC 32 does when it compiles a program, is to clear all debugging settings.   when it is executed in the **Gfa_OnRun** event sub. Before executing a project (Run) all internal debug settings of the IDE are reset to default values, then **Gfa_OnRun** is invoked. To start examining from the first line, the Tron procedure must be set after this initializing process.

To do something useful in tron procedure, the same functionality is available as when the **Tron** command was used in the source code itself.

TraceLnr holds the number of the line which will be executed next.

Trace$ is a string variable containing the line which will be executed next.

EdShowLine lnr% displays the debug arrow before the specified line. Normally, this is combined with TraceLnr:

EdShowLine TraceLnr

SrcCode$(lnr%) returns a string with the source code text of the specified line.

A program can only be inspected (trace on) when each statement is preceded by a call to the debug handler. These calls are inserted by default when a program is compiled to be run, except after **$StepOff** or when a subroutine is marked **Naked**. These calls are never inserted in external compiled modules (exe, gll, lg32).

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Declaring Variables

To declare a variable is to tell the program about it in advance. You declare a variable with the **Dim** statement, supplying a name for the variable:

Dim varname [**As** type] [= Value]

Variables declared with the **Dim** statement within a procedure exist only as long as the procedure is executing. When the procedure finishes, the value of the variable disappears. In addition, the value of a variable in a procedure is local to that procedure - that is, you can't access a variable in one procedure from another procedure. These characteristics allow you to use the same variable names in different procedures without worrying about conflicts or accidental changes.

Names of variables, constants, procedures etc are made up of letters (as well as the umlauts and the accented characters), underscore _, and digits, but not at the beginning of a name.

The optional **As** *type* clause in the **Dim** statement allows you to define the data type or object type of the variable you are declaring. Data types define the type of information the variable stores. Some examples of data types include **String**, **Integer**, and **Currency**.

Examples of GFA-BASIC 32 object types, include **Object**, **Form**, and **TextBox**.

There are other ways to declare variables. Declaring a variable using the **Global** or **Public** keyword makes it available throughout your application. For instance

Global Dim x As Long

Public x&, y$, a **As Double** = 1.0

Global String str1, str2, g%

Name the data type first, force g to Int.

Variable scope

GFA-BASIC 32 programs consist of a main part and subroutines. Any variable declared in the main part is a global variable, unless the declaration is preceded with the Local keyword. Declaring a variable with Dim in the main part does not restrict its scope to the main section, but is visible inside procedures as well. To make a variable local to the main part use:

Local [Dim] variable

This is also true for **Const** variables, to use a constant locally in the main part the **Const** keyword must be preceded by Local.

Local Const name = value

Variables declared with **Dim** in procedures and functions are local by default. To make a variable visible outside the function use Global or Public in front of Dim.

Global [**Dim**] str1$

If the global variable is initialized when declared in subroutine, the code to set the contents of the variable is not executed before the subroutine is executed. Declaring the variable public or global only tells the compiler to accept the variable as a global name. It does assign the value at runtime. (To assign a value to a global variable use Global Const.)

Static variables are global variables that are visible only in the procedure they are declared in.

Static String str1 = "Initial Value", str2

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Constants and Enumerations

Constants are a way to use meaningful names in place of a value that does not change. Constants store values that, as the name implies, remain constant throughout the execution of an application. You can use constants to provide meaningful names, instead of numbers, making your code more readable.

Enumerations provide a convenient way to work with sets of related constants, and to associate constant values with names. For example, you can declare an enumeration for a set of integer constants associated with the days of the week, and then use the names of the days rather than their integer values in your code.

Constants

[**Global**] **Const** name [**As** type] = value

Constants declared without Global or Public are local when used inside a procedure and global when used in the main part. Global constants can be declared anywhere in the program and are public to the entire program. Const and Dim are the same, except that Const prevents the variable to be changed.

If you do not specify a type, the constant takes the data type from the value. If you specify both type and initializer, the data type of initializer must be convertible to type. If neither data type nor initializer is present, the data type defaults to a 32-bits integer (Int, Integer, Int32, and Long).

```
Const WM_USER = 0x400  ' a 32-bits constant
```

In the next example the constant takes the data type String.

```
Const GFA = "Basic"  ' a string constant
```

Or, more explicitly

```
Const GFA As String = "Basic"
```

As data type are all intrinsic GFA-BASIC 32 data types allowed.

```
Const Updated = #12.07.1996#    ' becomes a Date
  data type
Const pi1 = 3.14                ' becomes a Double
  data type
Const pi2 = 3.14!               ' becomes a Single
  data type
Const currV = 15.20@            ' @ forces a
  Currency value
Const key As Short = $10        ' declared as
  short
```

You can use an expression to be assigned to the constant. The expression can be any combination of literals, other constants that are already defined, and enumeration members that are already defined. You can use arithmetic and logical operators to combine such elements. You cannot use variables or user defined functions in initializer. However, you can use conversion functions such as CByte and CShort, and GFA-BASIC 32 intrinsic functions.

```
Const WM_QUIT = WM_CLOSE + 2    ' 32-bits
Const Pi2     = PI / 2          ' Double
Const PiViertel = Atn(1)        ' Double,
  intrinsic function Atn
```

More than one constant can be listed

```
Const WM_USER = 0x400, WM_PAINT = 15, WM_CLOSE =
  $10, WM_QUIT = WM_CLOSE + 2
```

With the types Short=Word=Int16, Card and Byte=Int8 GFA-BASIC 32 performs an overflow check at compile time. Note When no type is specified GFA-BASIC 32 looks for the best fitting type starting with Integer, followed by Large, and when the value is still too large, a Double.

Enumerations

GFA-BASIC 32 lets you create enumerations. The use of enumerations can simplify certain programming tasks and make your program code easier to read. You create an enumeration with the Enum keyword. The constants are automatically assigned numerical values in order, starting with 0.

[**Global**] **Enum** name [= value] [,name [= value]]

```
Public Enum flVanilla, flChocolate, flCoffee,
  flStrawberry
```

This results in the constant flVanilla being equal to 0, flClocolate being equal to 1, and so on. Usually the actual numerical values of the constants in an enumeration do not matter, but if you want to assign specific values you can:

```
Enum WM_NULL, WM_CREATE, WM_DESTROY, WM_MOVE,
  WM_SIZE = 5, WM_ACTIVATE, WM_SETFOCUS,
  WM_KILLFOCUS, _
  WM_ENABLE = $A, WM_SETREDRAW, WM_SETTEXT,
    WM_GETTEXT, WM_GETTEXTLENGTH, WM_PAINT,
    WM_CLOSE, _
  WM_QUERYENDSESSION, WM_QUIT, WM_QUERYOPEN,
    WM_ERASEBKGND, WM_SYSCOLORCHANGE,
    WM_ENDSESSION, _
  WM_SHOWWINDOW = $18, WM_WININICHANGE = $1a
```

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

Literals

Invariant program elements are called "literals" or "constants." The terms "literal" and "constant" are used interchangeably here. Literals fall into 5 major categories: integer, floating-point (Single and Double), currency, date, and string. It's not necessary to declare constants explicitly with a data type, although typed code is easier to read and maintain than un typed code.

GFA-BASIC 32 uses the type of the expression used to initialize the constant. A numeric integer literal is cast by default to the Integer (32-bit) data type. The default data type for floating-point numbers is Double, and the keywords True and False specify a Boolean constant.

String Literals

Numeric Literals

Date and time literals

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Date and Time literals

The compiler treats literals enclosed within number signs (# #) as **Date**. (#5.7.95# or #5.7.95 12:42:30#). GFA-BASIC 32 provides three predefined formats for date/time literals. These are not country dependent so a program can be used in different languages.

#dd.mm.yyyy#

#mm/dd/yyyy#

#yyyy-mm-dd#

The recognition of the proper format Day.Month.Year, Month/Day/Year, or Year-Month-Day depends on the separation mark. The **Val**()-function accepts all three formats, as well.

These formats are independent of your locale and your computer's date and time format settings. The reason for this restriction is that the meaning of your code should never change depending on the locale in which your application is running. Suppose you hard-code a Date literal of #3/4/1998# and intend it to mean March 4, 1998. In a locale that uses mm/dd/yyyy, 3/4/1998 compiles as you intend. But suppose you pass the code on the users in other countries. In a locale that uses dd/mm/yyyy, your hard-coded literal would compile to April 3, 1998. In a locale that uses yyyy/mm/dd, the literal would be invalid (April 1998, 0003) and cause a compiler error.

Despite the above, GFA-BASIC 32 allows date literals in other locales by using CDate:

```
Dim d As Date = CDate("22 Nov 1995")
```

A **Date** type is internally interpreted as a **Double** type, except with explicit or implicit conversion to string routines (Print, Str, and Format). You can perform calculations on date/time values. Adding or subtracting integers adds or subtracts whole days; adding or subtracting fractions adds or subtracts fractions of days (expressed in hours and minutes). Thus, adding 20, adds 20 days, and subtracting 1/24 subtracts one hour.

```
Print Date + 8            // Returns the date in 8
 days
Print #24.12.2008# + 8
Print DmyToDate(24, 12, Year(Date)) - Date      //
 Returns the remaining days to Christmas eve
Print DateSerial((Year(Date), 12, 24, )) - Date
```

Because the current year is appended automatically...

```
#24.12#
```

...is automatically converted by the IDE to the 24th December of the current year.

Valid date values range from -647,434 (January 1, 100 A.D.) to 2,958,465 (December 31, 9999 A.D.). A date value of 0 represents December 30, 1899. Dates prior to December 30, 1899 are stored as negative numbers. Valid time values range from .0 (00:00:00) to .99999 (23:59:59). The numeric value represents a fraction of one day. You can convert the numeric value into hours, minutes, and seconds by multiplying the numeric value by 24.

Since GFA-BASIC 32 tries to maintain compatibility to VB, even when it is erroneous, you should use DateAdd and

DateDiff- functions when using dates before December 1899.

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

Numeric literals

A numeric value consists primarily of the digits 0 through 9 and a decimal point. Negative values need a leading minus sign (-); a plus sign (+) is optional for positive values.

If the value is an integer too large to fit in a 32-bit integer it will be stored in **Large** (64-bit) integer. GFA-BASIC 32 inserts the keyword **Large** before the literal (see examples). If the integer value is too large for a 64-bit integer the value is widened to a **Double**.

If the value contains a decimal point or is specified in the exponential format (e+-n) the value is assumed to be **Double**.

You can also force a literal value to be stored with a given precision by following the constant with one of the variable type-specifiers **#, !, @**

A **Double** value can be represented by appending **#** (3.14#, 3# ...).

A **Single** data type constant can be specified using **!** (3.14!, 3!, ...).

A **Currency** data type constant is formatted by appending a **@** (19.99@). Constants with a '@' prefix are not interpreted as octal, but as Currency constant.

```
Const DefaultInteger = 100          ' Default is
  Integer.
Const DefaultDouble = 54.3342       ' Default is
  Double.
Const MyString = "a"
```

```
Global Const MyDate = #01/15/2001#    ' Date
  constant
Global Const MyTime = #01:15:59#
Global Const MyLarge = Large 45     ' Forces data
  type to be a Large.
Global Const MySingle = 45.55!      ' Forces to be
  a Single.
```

Hexadecimal, octal and binairy literals

The compiler normally constructs an integer literal to be in the decimal (base 10) number system. You can force an integer literal to be hexadecimal (base 16) with the &H prefix, and you can force it to be octal (base 8) with the &O prefix. The digits following the prefix must be appropriate for the number system. The following table illustrates this.

Format	Number system
$nnn	Hexadecimal n=[0-9a-fA-F]
&Hnnn	Hexadecimal n=[0-9A-F] (VB compatible)
&nnn	Hexadecimal l n=[0-9A-F]
0xnnn	Hexadecimal l n=[0-9A-F] (C/C++ compatible)
&Onnn	Octal n=[0-7] (VB compatible)
0onnn	Octal n=[0-7] (starts with digit 0 followed with a letter o or O)
0bnnn	Binary n=[0-1] (starts with digit 0 followed with a letter b or B)
%nnn	Binary n=[0-1]
&Xnnn	Binary n=[0-1]
&X:nnn	The X represents a number base [1-9A-Z]. When X= 1 (&1:nn) the following n's are interpreted as being binary. When X= 7 (&7:nn) the value is octal. When X= 9 (&9:nn) stands for decimal and &F:nn for hexadecimal.

When X = Z (&Z:nn) the number has a base as 36 (10 digits + 26 letters)

Use **Base$**() with this unusual format. **Base$**(26467760 :Z) returns FRANK. The number base with **Base$** can be specified using :Z, but with 36 as well.
Base$**(&Z:FRANK, 10) returns 26467760. **Base$() is case insensitive.

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

String Literals

You must enclose a String literal within quotation marks (" "). If you need to include a quotation mark as one of the characters in the string, you use two contiguous quotation marks (""), for example

```
a$ = "1""2"         //  1"2
a$ = "1" + Chr$(34) + "2" // The same: 1"2
```

Literal strings separated by spaces are treated as one string:

```
a$ = "1234"    "5678"// a$ = "12345678"
```

Single characters can be specified using their ANSI-Code with # :

```
a$ = "This ia a test" #13#10 "Line 2" #13#10#0
// This ia a test
// Line 2
a$ = "This is a test" + Chr$(13, 10) + "Line 2" +
  Chr$(13, 10, 0)
a$ = "This is a test" + Chr$(13) + Chr$(10) +
  "Line 2" + _
  Chr$(13) + Chr$(10) + Chr$(0)
a$ = "This is a multiple line text" #13 #10 _
  "This is line 2" #13 #10 #0
```

The ANSI code specification is not limited to decimal values, for instance you can use hexadecimal values:

```
CrLf$ = #x0A #x0D
CrLf$ = #$A #$D
```

In addition characters may be specified in octal and binary values:

```
ChfromOctal$ = #o33
```

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Data Types

GFA-BASIC 32 supplies several numeric data types for handling numbers in various representations. Integral types represent only whole numbers (positive, negative, and zero), and non-integral types represent numbers with both integer and fractional parts.

Integral data types are those that represent only numbers without fractional parts.

The signed integral data types are the **Short** data type (16-bit), Integer data type (32-bit), and **Large** data type (64-bit). If a variable always stores integers rather than fractional numbers, declare it as one of these types.

The unsigned integral types are **Byte** (8-bit) and **Card** (16-bit).

Double(#)Double precision floating-point data type, 64 bit = 8 bytes with at least 15 digits of precision.

Minimum: **_minDbl** (-1.79769313486232e+308)

Maximum: **_maxDbl** (1.79769313486232e+308)

Epsilon: **_eps** or **_epsDbl** (2.22044604925031e-016)

Decadal Epsilon: **_eps10** or **_epsDbl10** (1.0e-014)

Smallest value: **_smallDbl** (2.2250738585072e-308)

Smallest value: **_tinyDbl** (4.94065645841247e-324)

Double is the most efficient of the fractional data types, because the processors on current platforms perform floating-point operations in double precision. However, operations with Double are not as fast as with the integral types such as Integer.

Single(!)Single precision floating-point , 32 bit = 4 bytes, with at least 7 digits of precision.

Minimum: **_minSng** (-3.402823e+038!)

Maximum: **_maxSng** (3.402823e+038!)

Epsilon: **_epsSng** (1.192093e-007!)

Decadal Epsilon: **_epsSng10** (1.0e-006!)

Smallest value: **_smallSng** (2.350989e-038!)

Smallest value: **_tinySng** (1.401298e-045!)

Integer (%),Integral value (Integer), 32 bits = 4 byte

Integer32,Range: -2147483648 to 2147483647

Int32, **Int,**

LongArithmetic operations are faster with integral types than with other data types. They are fastest with the Integer types. With calculations GFA-BASIC 32 does not perform overflow checking. Obviously, _maxInt + _maxInt is wrong when the result is stored in 32-bit integer.

Register Int The register keyword specifies that the variable is to be stored in a machine register (either edi or esi). Dim name As Register Int declares *Varname* as a variable, to be stored in a processor register. More than 2 register variable declarations per procedure are not allowed, and only 32-bit integer variables can be placed in a register.

Note: Registers don't have memory addresses, so you cannot obtain the variable's location (VarPtr, V:). Also, register variables cannot be passed by reference (ByRef) to subs. In case of an error (Try-Catch, On Error GoTo Resume) the contents of the register variable is undefined.

Short(&),16 bit signed integral value.

Word, **Int16**Range: -32768 - 32767
Integer16

Card16 bit unsigned integral. Range: 0 - 65535

Byte(|)8 bit unsigned integral. Range: 0 - 255

Bool (?),False (0) or True (-1)

BooleanAssignments to a Boolean variable are stored as either 0 or -1.

LargeIntegral data type, 64 Bit=8 Byte. Integer64 Range: -9223372036854775808 to 9223372036854775807. Int64

Currency(@) fixed-point type, 8 bytes. Range: -922337203685477.5808 to 922337203685477.5807. The Currency data type supports up to four digits to the right of the decimal separator and fifteen digits to the left; it is an accurate fixed-point data type suitable for monetary calculations. Floating-point (Single and Double) numbers have much larger ranges than Currency, but can be subject to small rounding errors.

DateDate and time, 64 bit Double format. Range: #01.01.0100# to #31.12.9999 23.59.59#

HandleIdentification number (32 bit). **Null** = **CHandle**(0)

String ($)variable-length string

String * lenString with fixed length, maximum size 1 megabyte.

ObjectAutomation object data (IDispatch), 32 bit. A variable declared as Object is one that can subsequently be assigned (using the Set statement) to refer to any actual object recognized by the application.

VariantA Variant variable is capable of storing all system-defined types of data. You don't have to convert between these types of data if you assign them to a Variant variable; GFA-BASIC 32 automatically performs any necessary conversion.

Converting Data Types

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Boolean Data Type

Purpose

The Boolean type represents the two logical values, **True** and **False**.

Syntax

Dim *name* As **Boolean | Bool**

Dim *Name***?**

Description

Type declaration character is **?**. Range: -1 - 0.

False = 0

True = -1

Boolean values are stored in a Byte.

With an array of **Bool** the values are stored in a bit.

When other numeric types are converted to **Boolean** values, 0 becomes **False** and all other values become **True**. When **Boolean** values are converted to other data types, **False** becomes 0, and **True** becomes -1.

Example

```
Dim d As Boolean = -1
Local Bool d1 = True
Global d2?
```

```
Dim f?(7)    // occupies 1 Byte
Dim f2?(10) // occupies 2 bytes
```

Remarks

Note There is a compiler **bug** when setting the eighth Bool in a row of 8 Booleans to False. The entire byte containing the 8 Booleans is affected, because setting the eighth bit generates an '**and** a-byte, 8' assembler instruction.

```
Type BoolTrouble
  a0 As Bool
  a1 As Bool
  a2 As Bool
  a3 As Bool
  a4 As Bool
  a5 As Bool
  a6 As Bool
  a7 As Bool' <- 8th bool in a row
EndType
Dim bl As BoolTrouble
Message bl.a0
bl.a0 = True
Message bl.a0
bl.a7 = False  ' and V:bl.a0, 8
Message bl.a0
```

You have a few options. You could use the eighth bit as a dummy and don't use it. Or, you can explicitly set and clear the bit:

```
Bset bl.a7, 1' bl.a7 = True
Bclr bl.a7, 1' bl.a7 = False
```

See Also

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Byte Data Type

Purpose

A 8 bit unsigned integral value.

Syntax

Dim name As **Byte**

Dim name**|**

Description

Type declaration character is **|**.

Range: 0 - 255

Arithmetic operations are faster with integral types than with other data types. They are fastest with the 32-bit Integer types.

Example

```
Dim d As Byte = 2
Local Byte d1
Global d2|
```

Remarks

The unsigned integral types are **Byte** (8-bit) and **Card** (16-bit).

See Also

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Card Data Type

Purpose

A 16 bit unsigned integral value.

Syntax

Dim name1 As **Card**

Description

This data type does not have declaration character.

Range: 0 - 65535

Arithmetic operations are faster with integral types than with other data types. They are fastest with the 32-bit Integer types.

Example

```
Dim d As Card = 2
Local Card d1
```

Remarks

The unsigned integral types are **Byte** (8-bit) and **Card** (16-bit).

See Also

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Currency Data Type

Purpose

Fixed-point floating-point data type.

Example

Dim *name* As **Currency**
Dim *name*@

Description

Type declaration character is **@.**

Currency variables are stored as 64-bit (8-byte) numbers in an integer format, scaled by 10,000 to give a fixed-point number with 15 digits to the left of the decimal point and 4 digits to the right. This representation provides a range of:

-922337203685477.5808 to 922337203685477.5807

The **Currency** data type supports up to four digits to the right of the decimal separator and fifteen digits to the left; it is an accurate fixed-point data type suitable for monetary calculations. Floating-point (Single and Double) numbers have much larger ranges than **Currency**, but can be subject to small rounding errors.

A **Currency** occupies 64 bit = 8 bytes with at least 7 digits of precision.

Use the **Currency** data type instead of **Single** or **Double** for monetary values. If you specify more than four decimal places in a currency expression, GFA-BASIC 32 rounds to four places before evaluating the expression.

Example

```
Dim d As Currency = 2.10
Local Currency d1
Local d2@
Const DD = 1@
```

See Also

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Double Data Type

Purpose

Double precision floating-point data type.

Example

Dim name As **Double**
Dim name**#**

Description

Type declaration character is **#**.

Double is the most efficient of the fractional data types, because the processors on current platforms perform floating-point operations in double precision. However, operations with **Double** are not as fast as with the integral types such as Integer.

A **Double** occupies 64 bit = 8 bytes with at least 15 digits of precision.

Minimum: **_minDbl** (-1.79769313486232e+308)

Maximum: **_maxDbl** (1.79769313486232e+308)

Epsilon: **_eps** or **_epsDbl** (2.22044604925031e-016)

Decades Epsilon: **_eps10** or **_epsDbl10** (1.0e-014)

Smallest value: **_smallDbl** = 2.2250738585072e-308)

Smallest value: **_tinyDbl** (4.94065645841247e-324)

Example

```
Dim d As Double = 2.10
Local Double d1, d2#
Const DD = 1#
Const DD_1 = _eps
```

See Also

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

Handle Data Type

Purpose

A 32-bit integral data type.

Example

Dim *name* As **Handle**

Description

A **Handle** occupies 32 bit (4 bytes). It doesn't have a type declaration character.

The **Handle** data type is meant to store values that identify an object; an indirect reference to an operating system resource. Often a handle is a number assigned to a window that is used by the operating system to keep track of the attributes of the window. But a handle can also be a pointer to memory, a number identifying an opened file, etc.

Although the **Handle** data type is a 32-bit integer, it cannot be used in arithmetic operations. This provides some security against writing. A **Handle** can be assigned to another data type, however that would undo its purpose.

In the same way, a value can be assigned to a **Handle**. A conversion is made using **CHandle**().

Example

```
Dim h As Handle
h = _File(# 1)
```

```
If h != Null
  h ++        // Error: Operation not allowed on
    Handle
EndIf
```

See Also

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Int16, Integer16, Word, Short Data Type

Purpose

A 16 bit signed integral value.

Syntax

Dim *name* As **Int16** | **Word** | **Short** | **Integer16**
Dim *name***&**

Description

Type declaration character is **&**.

Int16, **Integer16**, **Word**, and **Short** are keywords for a 16-bits signed integer.

Range: -32768 to 32767

Arithmetic operations are faster with integral types than with other data types. They are fastest with the 32-bit Integer types.

Example

```
Dim d As Word = 2
Local Short d1
Local d2&
```

See Also

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Int, Int32, Integer32, Long Data Type

Purpose

A 32 bit signed integral value.

Syntax

Dim *name* As **Int** | **Int32** | **Integer32** | **Long**
Dim *name***%**

Description

Type declaration character is **%**.

Int, Int32, Integer32, and **Long** are keywords for a 32-bits signed integer.

Range: -2147483648 to 2147483647

Defined constants:

_maxInt = 2147483647

_minInt = -2147483648

Arithmetic operations are faster with integral types than with other data types. They are fastest with the 32-bit Integer types.

Example

```
Dim d As Int = 2
```

```
Local Long d1, d2%
Const DD = 1
Const DD_1 = _maxInt
```

Remarks

With calculations GFA-BASIC 32 does not perform overflow checking. Obviously, **_maxInt** + **_maxInt** is wrong when the result is stored in 32-bit integer.

See Also

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Int64, Integer64, Large Data Type

Purpose

A 64 bit signed integral value.

Syntax

Dim *name* As **Int64** | **Integer64** | **Large**

Description

This type does not have type declaration character.

Int64, Integer64 and **Large** are keywords for a 64-bits signed integer.

Range: -9223372036854775808 to 9223372036854775807

Defined constants:

_maxLarge = 9223372036854775807

_minLarge = -9223372036854775808

Arithmetic operations are faster with integral types than with other data types. They are fastest with the 32-bit Integer types.

Example

```
Dim d As Large = 2
Local Int64 d1
```

See Also

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Object Data Type

Purpose

The **Object** data type is a 32-bit (4-byte) address that refer to COM objects within an application or within some other application.

Syntax

Dim name **As Object**

Description

When you create an application in GFA-BASIC 32, you work with objects. You can use objects provided by GFA-BASIC 32 - such as controls, forms, and data access objects. You can also control other applications' objects from within your GFA-BASIC 32 application.

Declaring an object variable with the **As Object** clause creates a variable that can contain a reference to any type of (OLE) object. However, access to the object through that variable is late bound; that is, the binding occurs when your program is run. To create an object variable that results in early binding, that is, binding when the program is compiled, declare the object variable with a specific class ID. For example, you can declare and create the following Microsoft Excel references:

```
Dim xlApp As Object
Dim xlBook As Object
Dim xlSheet As Object
Set xlApp = CreateObject("Excel.Application")
Set xlBook = xlApp.Workbooks.Add
Set xlSheet = xlBook.Worksheets(1)
```

After you declare an object variable, you must assign an object reference to the variable before you can use the object's properties, methods, and events. You can assign a reference to a new object in a **Set** statement by using the **CreateObject** or **GetObject** function.

The **Object** data type stores a pointer to an IDispatch interface, the late binding mechanism of COM. When a COM object provides an IDispatch interface, the properties and methods can be executed through a standard function called *Invoke*. Rather than executing a property or method directly, as with early binding, the *Invoke* function takes numerous parameters describing the property or method to call, the possible parameters converted to Variants, an exception info block for returning error information, and some more. *Invoke* itself must lookup the name of the property or method in the COM library and then call it by its address. Calling *Invoke* for a property or method is a time consuming process, therefore.

Example

```
OpenW 1
Dim oForm As Object
Set oForm = Win_1.Object
Win_1.AutoRedraw = 1  ' Fast
oForm.AutoRedraw = 1  ' Slow
Do
  Sleep
Until Me Is Nothing

Sub Win_1_OnCtrlHelp(Ctrl As Object, x%, y%)
  ' IDispatch reference to the control.
  Print Ctrl.WhatsThisHelpID // Slow
EndSub
```

See Also

Set, CreateObject, GetObject

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Single Data Type

Purpose

Single precision floating-point data type.

Example

Dim name As **Single**
Dim name**!**

Description

Type declaration character is **!**.

A **Single** occupies 32 bit = 4 bytes with at least 7 digits of precision.

Minimum: **_minSng** (-3.402823e+038!)

Maximum: **_maxSng** (3.402823e+038!)

Epsilon: **_epsSng** = 1.192093e-007!)

Decades Epsilon: **_epsSng10** (1.0e-006!)

Smallest value: **_smallSng** = 2.350989e-038!)

Smallest value: **_tinySng** (1.401298e-045!)

Example

```
Dim d As Single = 2.10
Local Single d1, d2!
Const DD = 1!
Const DD_1 = _epsSng
```

See Also

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

String Data Type

Purpose

A **String** variable consists of a sequence of 8-bit characters. There are two kinds of strings: variable-length (technically capable of storing 2 billion (2^{31}) characters but are generally limited to ±256 million (2^{28})) and fixed-length strings (±1 million (2^{20} - 1) characters).

Syntax

Dim varname **As String** [= *string-literal*] (variable-length)

Dim varname **As String** * *len* (fixed-length)

Dim varname**$**

Description

The codes for **String** characters range from 0-255. The first 128 characters (0-127) of the character set correspond to the letters and symbols on a standard U.S. keyboard. These first 128 characters are the same as those defined by the ASCII character set. The second 128 characters (128-255) represent special characters, such as letters in international alphabets, accents, currency symbols, and fractions and can differ depending on the value of **Mode(Lang)** and the current font face.

The type-declaration character for a variable length **String** is the dollar sign (**$**).

Variable-length strings

A variable-length string has a 4 byte pointer (*descriptor*) to a dynamically allocated memory block containing the string characters; the four bytes prior to this block store a 32-bit value containing the length of the string. For an empty string this pointer is null (0), meaning that no memory is allocated. Assigning data to a string will (re)allocate memory for the string data. Each string is terminated with a null character = Chr(0), but the terminating null is not counted as part of the string. The total size of the allocated string memory is 4 (length) + [*the string itself*] + 1 (#0). Given a string s$, the following is true:

String descriptor: **ArrPtr**(s$) or ***s$**

String memory: **VarPtr**(s$) or **V:** s$ (calculated from the descriptor: **{***s$**}**)

Length: **Len**(s$) (using the descriptor *length* = **{{***s$**}** - 4**},** or using the address *length* = **{V:** s$ - 4**}**)

Fixed-length string

A fixed-length string is a piece of memory used to store a string and is the only string variable type allowed within a **Type** declaration (excepting strings in variants). This type of string does not have a descriptor, the variable directly addressing the memory allocated using the declaration statement **String***n* (fixed strings can not use the string literal ($)). The fixed-string is initialized with null characters; if a smaller string is assigned to a fixed-string, spaces will be added to the end of the string. However, if a larger string is assigned, only the characters which fit into the length of the string will be stored and any remaining characters will be lost.

A fixed string is not terminated with a null character (Chr(0)) and it has no length data field in front of it. The length of the fixed-string is inserted at compile time and not calculated at run time. The **VarPtr** and **V:** functions return the address of its memory location while the **ArrPtr** (or *****) function returns the first four bytes of the fixed string; hence, these functions have no practical meaning for a fixed-string.

String literal

A constant string (string-literal) is a sequence of characters surrounded by quotation marks (").

```
Dim sName As String = "Basic"
Const BASICNAME As String = "GFA-BASIC"
```

To place a quotation mark (") inside a string constant, you must either place two quotes together or build a string using the character code for a quote (34) either using **Chr**(34) or #34:

```
sName = "A ""quoted"" constant."  // sName now
 holds: A "quoted" constant
sName = "A " & Chr$(34) & "quoted" #34 "
 constant."
```

Strings in Variants

Strings stored in Variants are BSTR types in UNICODE or 'wide character' format. GFA-BASIC 32 takes care of allocating and converting the ANSI string to UNICODE by using a faster variant of the *MultiByteToWideChar* API function that maps a character string to a wide-character (Unicode) string. Some GFA-BASIC 32 string functions can be used on strings in Variants, providing support for UNICODE strings, although others, such as **len** do not.

A BSTR is more than a pointer to Unicode characters. The string length is maintained in a long variable just before the start address being pointed to, and the string always has an extra null character after the last character of the string. This null isn't part of the string, and you may have additional nulls embedded in the string. The BSTR data type is allocated using OLE Automation String functions, like *SysAllocString*.

The **VarPtr** and **V:** functions return the address of the **Variant** variable, not the string. To find the string data use: StrPtr% = { V:Variant + 8}.

OLE Strings

Many OCX and Automation objects take a string value as a parameter. These strings are always BSTRs. When GFA-BASIC 32 comes to a point that it must pass or assign an ANSI string to a COM object, it converts the string to a BSTR first and passes the BSTR to the OLE property or method.

The reverse is true also. When a string returned from a COM object is assigned to a String data type, the BSTR is converted to ANSI using the internal GFA-BASIC 32 function mentioned above.

Strings in API functions

The way strings are passed to Windows API functions depend on how the external functions are declared. For the 1000 or so built-in (ANSI) API functions, the function arguments are not type checked at compile time. The compiler is only aware of the number of parameters and accepts 32-bits values only. Those API functions that expect a (pointer to a) string must be provided with the address of the GFA-BASIC 32 string using **V:**. For instance, the built-in *CharLower* function converts a character string or a single character to lowercase. The function takes only one parameter: a LPTSTR pointer to a null-terminated string. The string is converted in place, so that the return value is equal to the passed value. The following code does the job:

```
Dim s$ = "GFA-BASIC 32"
Debug V:s$, CharLower(V:s$), s$   // [address1]
  [address1] gfa-basic 32
```

What happens when *s$* is passed, instead of its address? GFA-BASIC 32 pulls in one of 32 string buffers of 1030 bytes and copies the contents of s$ to the temporary buffer and passes the address of the buffer to the API function. The string is converted in place and thus the temporary buffer is modified and the memory location of the buffer is returned. *s$* remains unchanged.

```
Dim s$ = "GFA-BASIC 32"
Debug V:s$, CharLower(s$), s$ //  [address1]
  [address2] GFA-BASIC 32
```

By using **Declare** the API function can be introduced to GFA-BASIC 32 and force type checking on the parameters. A string parameter must always be declared using **ByVal** to get its address (**V:**) passed to the API function. A **ByRef** string parameter would obtain the address of the descriptor (**ArrPtr**).

```
Dim s$ = "GFA-BASIC 32"
Debug V:s$, CharLowerA(s$), s$
Declare Function CharLowerA Lib "user32" (ByVal
  lpsz As String) As Long
```

```
// [address1] [address1] gfa-basic 32
```

Remarks

The case functions **UCase** and **LCase** are ASCII functions. The **Upper** and **Lower** functions convert the second 128 characters (128-255) also.

All string functions come in two versions, one with an ending **$** type declaration character and one without. In contrast with VB, both version return a string data type. In VB the function without the **$** character returns a Variant.

See Also

Declare, UCase, LCase, Upper, Lower, Variant, ArrPtr, VarPtr, V:, Left$, Right$, Mid, Mid$, SubStr, InStr, RinStr, Mirror

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Variant Type

Purpose

The **Variant** data type is the data type for all variables that are not explicitly declared as some other type (using statements such as **Dim**, **Local**, **Global**, **Public**, or **Static**). The **Variant** data type has no type-declaration character.

Syntax

Dim v **As Variant**

v:variable name

Description

A **Variant** is a special data type that can contain any kind of data except fixed-length String data. (**Variant** types don't support user-defined types.) A **Variant** can also contain the special values **Empty**, **Error**, **Missing**, **Nothing**, and **Null**. You can determine how the data in a **Variant** is treated using the VarType and TypeName functions.

Internal Representation of Values in Variants

Numeric Values Stored in Variants

Strings Stored in Variants

Date/Time Values Stored in Variants

Objects Stored in Variants

The Empty Value

The Missing Value

The Null Value

Variant Error Types

Supported Variable Types

A **Variant** cannot store a **Large** data type. When assigned, a **Large** is converted to a **Double.**

A **Handle** is stored as a **Long** (VT_I4).

You can use the **Variant** data type in place of any data type to work with data in a more flexible way. If the contents of a **Variant** variable are digits, they may be either the string representation of the digits or their actual value, depending on the context. For example:

```
Dim Var As Variant = 98052
```

In the preceding example, *Var* contains a numeric representation-the actual value 98052. Arithmetic operators work as expected on **Variant** variables that contain numeric values or string data that can be interpreted as numbers. If you use the **+** operator to add *Var* to another **Variant** containing a number or to a variable of a numeric type, the result is an arithmetic sum.

Example

```
Dim v = Null ' declares and initializes a Variant
Dim va(1 .. 3)' declares a Variant array
```

Known Issues

There is an odd bug when passing Boolean values to an optional variant parameter in a function IF the function is called form a procedure containing a Gosub...Return structure - an Access Violation Error is returned for no apparent reason pointing to the line containing `Return`. This is illustrated by the code examples below:

```
trial

Procedure trial
  Local enb As Boolean = True
  VarTrial(10, enb)
  GoSub Here
Return
  Here:
  Print "Go to here"
EndProcedure

Function VarTrial(a%, Optional v As Variant)
  Print a, v
EndFunction
```

This is an error within the compiler and, currently, unfixable. If you experience this, simple workarounds are: use a different varaible type in the calling procedure (anything but Boolean seems to work); or change the optional parameter in the called Function to type Boolean.
[Reported by James Gaite, 11/03/2018]

Remarks

Variants can be used very easily, due to their high flexibility, but with a loss of performance. A counting loop...

```
Local a As Variant
For a = 1 To 100
Next a
```

...will be many times slower than the corresponding one with an Integer loop:

```
Local a As Int
For a = 1 To 100
Next a
```

It should also be npted that automatic conversion of data to a **Variant** does have its limits. What should be done when two Variants are added (or concatenated) when one contains a string and the other a numeric value? So, what should the following mean:

vntC **= CVar(**"123"**) + CVar(**456**)**

1 - Add them as they were both numeric values, so convert the string to a numeric value.

2 - Concatenate them as strings, resulting in the string "123456".

3 - None of the above, but generates an error.

GFA-BASIC 32 performs as VB and takes option 1.

{Created by Sjouke Hamstra; Last updated: 13/03/2018 by James Gaite}

Internal Representation of Values in Variants

Variant variables maintain an internal representation of the values that they store. This representation determines how GFA-BASIC 32 treats these values when performing comparisons and other operations. When you assign a value to a **Variant** variable, GFA-BASIC 32 uses the most compact representation that accurately records the value. Later operations may cause GFA-BASIC 32 to change the representation it is using for a particular variable. (A **Variant** variable is not a variable with no type; rather, it is a variable that can freely change its type.)

A variant always takes up 16 bytes, no matter what you store in it. The first two bytes store the information of the current data or variable type stored in the Variant, while the last eight bytes either store the data value or, in the case of Objects, strings, and arrays which are not physically stored in the **Variant**, four of these eight bytes are used to hold either an object reference, or a pointer to the string or array, with the actual data being stored elsewhere.

Most of the time, you don't have to be concerned with what internal representation GFA-BASIC 32 is using for a particular variable as GFA-BASIC 32 handles conversions automatically. If you want to know what value GFA-BASIC 32 is using, however, you can use the **VarType** function.

For example, if you store values with decimal fractions in a **Variant** variable, GFA-BASIC 32 always uses the **Double** internal representation. If you know that your application does not need the high accuracy (and slower speed) that a

Double value supplies, you can speed your calculations by converting the values to **Single**, or even to **Currency**:

```
If VarType(X) = 5 Then X = CSng(X)    ' Convert to
  Single
```

With an array variable, the value of **VarType** is the sum of the array and data type return values. For example, this array contains **Double** values:

```
Sub Form_Click()
  Dim dblSample(2) As Double
  MsgBox VarType(dblSample)
End Sub
```

Variant

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

Numeric Values Stored in Variants

When you store whole numbers in **Variant** variables, GFA-BASIC 32 uses the most compact representation possible. For example, if you store a small number without a decimal fraction, the **Variant** uses an **Integer** representation for the value. If you then assign a larger number or a number with a fractional component, a **Double** value.

Sometimes you want to use a specific representation for a number. For example, you might want a **Variant** variable to store a numeric value as **Currency** to avoid round-off errors in later calculations. GFA-BASIC 32 provides several conversion functions that you can use to convert values into a specific type (see "Converting Data Types" earlier in this chapter). To convert a value to **Currency**, for example, you use the **CCur** function:

```
PayPerWeek = CCur(hours * hourlyPay)
```

An error occurs if you attempt to perform a mathematical operation or function on a **Variant** that does not contain a number or something that can be interpreted as a number. For example, you cannot perform any arithmetic operations on the value U2 even though it contains a numeric character, because the entire value is not a valid number. Likewise, you cannot perform any calculations on the value 1040EZ; however, you can perform calculations on the values +10 or -1.7E6 because they are valid numbers. For this reason, you often want to determine if a **Variant** variable contains a value that can be used as a number. The **IsNumeric** function performs this task:

```
Local anyNumber
Do
 anyNumber = InputBox("Enter a number")
Loop Until IsNumeric(anyNumber)
MsgBox "The square root is: " & Sqr(anyNumber)
```

When GFA-BASIC 32 converts a representation that is not numeric (such as a string containing a number) to a numeric value, it uses the Regional settings (specified in the Windows Control Panel) to interpret the thousands separator, decimal separator, and currency symbol.

Thus, if the country setting in the Windows Control Panel is set to United States, Canada, or Australia, these two statements would return true:

```
Print IsNumeric("$100")
Print IsNumeric("1,560.50")
```

While these two statements would return false:

```
Print IsNumeric("DM100")
Print IsNumeric("1.560,50")
```

However, the reverse would be the case - the first two would return false and the second two true - if the country setting in the Windows Control Panel was set to Germany.

If you assign a **Variant** containing a number to a string variable or property, GFA-BASIC 32 converts the representation of the number to a string automatically. If you want to explicitly convert a number to a string, use the **CStr** function. You can also use the **Format** function to convert a number to a string that includes formatting such as currency, thousands separator, and decimal separator symbols. The **Format** function automatically uses the appropriate symbols according to the Regional Settings Properties dialog box in the Windows Control Panel.

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Strings Stored in Variants

Strings are stored in the BSTR format within a variant (actually the BSTR is stored separately and referenced from the Variant) which is a version of 16-bit Unicode format, while GFA-BASIC 32 stores strings as 8-bit characters (GFA-BASIC 32 is based on 8-bit strings because Windows 95 didn't support Unicode (16 bit) functions). Therefore, each time a string is assigned to a **Variant** or vice versa, a conversion is performed automatically by GFA-BASIC32, as shown in the example below:

```
Local a$, vnt As Variant
a$ = "String"    // The word 'String' is stored in
  8-bit format in the GFA string data type
vnt = a$         // GFA converts the 8-bit string
  to BSTR 16-bit format and assigns it to a Variant
Print vnt        // GFA then converts the value in
  the BSTR back to 8-bit format before 'Print'-ing
  to screen
```

Due to this automatic conversion performed by GFA-BASIC32, strings in a Variant can use most of the commands and functions designed for the 8-bit GFA-BASIC32 string data type, as shown below:

```
Local a$ = "String" , vnt As Variant = a$
Print Len(a$), Len(vnt)
Print Mid(a$, 2, 2), Mid(vnt, 2, 2)
Print Mirror$(a$), Mirror$(vnt)
Print Upper(a$), Upper(vnt)
```

An example of a keyword that does not work with strings in a Variant is the Mid$ command (not to be confused with the **Mid** function shown above).

Generally, storing and using strings in **Variant** variables poses few problems. However, sometimes the result of the **+** operator can be ambiguous when used with two **Variant** values. If both of the **Variants** contain numbers, the **+** operator performs addition. If both of the **Variants** contain strings, then the **+** operator performs string concatenation. But if one of the values is represented as a number and the other is represented as a string, the situation becomes more complicated. GFA-BASIC 32 first attempts to convert the string into a number. If the conversion is successful, the **+** operator adds the two values; if unsuccessful, it generates a Type mismatch error.

To make sure that concatenation occurs, regardless of the representation of the value in the variables, use the **&** operator. For example,

```
Form_Click

Sub Form_Click ()
  Dim X, Y
  X = "6"
  Y = "7"
  Print X + Y, X & Y    // 67       67
  X = 6
  Print X + Y, X & Y    // 13       67
End Sub
```

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

Date/Time Values Stored in Variants

Variant variables can also contain date/time values. Several functions return date/time values. For example, **DateSerial** can be used to return the number of days left until a particular day in the year:

```
Dim xmas, rightnow, daysleft, hoursleft,
 minutesleft ' As Variant by default
rightnow =  Now   ' Now returns the current
 date/time.
xmas = DateSerial((Year(rightnow) +
 Iif(Month(rightnow) = 12 And Day(rightnow) > 24,
 1, 0), 12, 25,))
daysleft = Int(xmas - rightnow)
hoursleft = 24 - Hour(rightnow)
minutesleft = 60 - Minute(rightnow)
Print daysleft & " days, ";
Print hoursleft & " hours and ";
Print minutesleft & " minutes left until Christmas
 Day."
```

You can also perform math on date/time values. Adding or subtracting integers adds or subtracts days; adding or subtracting fractions adds or subtracts time. Therefore, adding 20, adds 20 days, while subtracting 1/24 subtracts one hour.

The range for dates stored in **Variant** variables is January 1, 0100, to December 31, 9999. Calculations on dates don't take into account the calendar revisions prior to the switch to the Gregorian calendar, however, so calculations producing date values earlier than the year in which the

Gregorian calendar was adopted (1752 in Britain and its colonies at that time; earlier or later in other countries) will be incorrect.

You can use date/time literals in your code by enclosing them with the number sign (#), in the same way you enclose string literals with double quotation marks (""). For example, you can compare a **Variant** containing a date/time value with a literal date:

```
If SomeDate > #03/06/1993# Then
```

Similarly, you can compare a date/time value with a complete date/time literal:

```
If SomeDate > #03/06/1993 13:20:00# Then
```

If you do not include a time in a date/time literal, GFA-BASIC 32 sets the time part of the value to midnight (the start of the day).

GFA-BASIC 32 accepts a wide variety of date and time formats in string-based literals as well (although not in true literal form surrounded by #s). These are all valid date/time values:

```
Print CDate("3-6-93 13:20")
Print CDate("March 27, 1993 1:20am")
Print CDate("Apr-2-93")
Print CDate("4 April 1993")
```

In the same way that you can use the **IsNumeric** function to determine if a **Variant** variable contains a value that can be considered a valid numeric value, you can use the **IsDate** function to determine if a **Variant** contains a value that can be considered a valid date/time value. You can then use the **CDate** function to convert the value into a date/time value.

For example, the following code tests the Text property of a text box with **IsDate**. If the property contains text that can be considered a valid date, GFA-BASIC 32 converts the text into a date and computes the days left until the end of the year:

```
Dim SomeDate, daysleft
Ocx Label lbl = "Enter Date:", 10, 10, 60, 14 :
  lbl.BackColor = RGB(255, 255, 255)
Ocx TextBox Text1 = "", 70, 9, 100, 14 :
  .BorderStyle = 1
Ocx Command cmd = "Calculate", 175, 7, 70, 18
Ocx Label Text2 = "", 10, 30, 200, 14 :
  Text2.BackColor = RGB(255, 255, 255)
Do : Sleep : Until Me Is Nothing

Sub cmd_Click
  If IsDate(Text1.Text) Then
    SomeDate = CDate(Text1.Text)
    daysleft = Int(DateSerial((Year(SomeDate) + _
      1, 1, 1, )) - SomeDate)
    Text2.Text = daysleft & " days left in the
      year."
  Else
    MsgBox Text1.Text & " is not a valid date."
  End If
EndSub

Sub Text1_KeyPress(Ascii&)
  Text2.Text = ""
EndSub
```

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

Objects Stored in Variants

Objects can be stored in **Variant** variables. This can be useful when you need to gracefully handle a variety of data types, including objects. For example, all the elements in an array must have the same data type. Setting the data type of an array to **Variant** allows you to store objects alongside other data types in an array.

The **IsObject**() function determines if a Variant holds an OCX or IDispatch type value.

```
Ocx TextBox txt = "TextBox", 10, 40, 100, 100
Local vnt As Variant = txt
Print "vnt = "; vnt
Print "Is vnt as Object? ... " & IsObject(vnt)
Do : Sleep : Until IsNothing(Me)
```

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

The Empty Value

Syntax

Variant = **Empty**
Boolean = **IsEmpty**(*Variant*)

Description

Generally, when a variable is created, its value is automatically initialised by GFA-BASIC32 (unlike C++) as a zero for numerical values or a zero-length string ("") for strings; however, when a **Variant** is created, it can not always be initialised according to its variable type as that may not be known until a value is assigned, so, instead, it is considered to be **Empty**.

As every **Variant** must technically have a value, a special **Empty** value is supported for **Variants** which have had no other value assigned. This special value has a VarType of 0 and is really just a block of 16 bytes all containing the value zero. One advantage to having this special value is that **Variants** which have previously been initialised with a value, and thus a variable type, can be 'uninitialised' by assigning it the **Empty** value. Furthermore, if an automation object held in a **Variant** is set to **Empty**, the automation object is set to **Nothing**.

When you use a **Variant** in an expression which is uninitialised and thus **Empty**, GFA-BASIC32 will substitute either 0 or a zero-length string, depending on the expression. Nevertheless, sometimes you may need to know if a **Variant** variable has been initialised since the variable was created and to do this, you can use the

IsEmpty function which will return TRUE if no value has been assigned.

The **Empty** value disappears as soon as any value is assigned to a **Variant** variable (including the value of 0, the zero-length string, and the **Null** value).

Example

```
Local vnt As Variant
Print "Is vnt Empty? ... "; IsEmpty(vnt)          //
 Confirms that vnt is Empty
Print "2 + vnt = "; 2 + vnt                        //
 If added to a number then GFA assumes a value of
 0
Print "'Hello' + vnt = "; "Hello" + vnt            //
 If added to a string then GFA assumes a zero-
 length string
If IsEmpty(vnt) Then vnt = 0
Print "Is vnt Empty? ... "; IsEmpty(vnt)          //
 Confirms that vnt is no longer Empty
vnt = Empty
Print "Is vnt Empty? ... "; IsEmpty(vnt)          //
 Confirms that vnt is once again Empty
```

Remarks

The **Empty** value should NOT be confused with either the **Null** value, which indicates that the **Variant** variable intentionally contains no valid data, or the **Missing** value, which is generally used to indicate that an optional Variant parameter to a function or procedure has not been passed.

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

The Missing Value

Syntax

Variant = **Missing**
Boolean = **IsMissing**(*String* | *Variant*)

Description

Generally, a **String** or **Variant** is marked as Missing if it is an optional parameter in a sub routine and no value is passed and this can be tested using the **IsMissing** function.

The **Missing** keyword is provided to allow a **Variant** (only) to be set to this state if so required.

Example

```
Print Test(), IsMissing(Test())
Print Test(14), IsMissing(Test(14))
Print Test(Missing), IsMissing(Test(Missing))

FunctionVar Test(Optional var)
  Test = var
EndFunction
```

Remarks

A **String** or **Variant** containing **Missing** has no value and will generally cause an error if used in a function or command. It should not be confused with Null or Empty.

In addition, as the value **Missing** is technically an Error Variant Type, IsError will also return TRUE if it is assigned to a Variant.

See Also

Empty, Nothing, Null

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

The Null Value

Purpose

The **Null** keyword is used with a:

1. Variant to indicate that it intentionally contains no valid data.
2. Handle data type to indicate a null handle.
3. API function to pass a null value for **ByRef** parameters.

Syntax

Variant | *Handle* = **Null**
Boolean = **IsNull**(*Variant* | *Handle*)

Description

Variant variables are not set to **Null** unless you explicitly assign **Null** to them, so if you don't use **Null** in your application, you don't have to write code that tests for and handles it. You can assign **Null** as follows:

```
Dim v As Variant = Null
```

You can use the variant function **IsNull** to test if a Variant variable contains **Null**:

```
If IsNull(variant) Then Print "Variant contains
 Null"
```

Data "#Null#" can be used with **Data** lines to initialize a Variant.

A handle data type can simply be compared with **Null**. Here **Null** is defined as **CHandle**(0).

```
If hWnd == Null Then Print "Handle is Null"
```

For API functions that have parameters declared as **ByRef**, the **Null** value may be passed (if that API function can handle a Null value), in contrast to the number 0.

Example

```
OpenW 1
Local a As Variant, b As Handle, x%
Print IsNull(b) // result True
a = ""
Print IsNull(a) // result 0
b = 2
Print IsNull(b) // result False
b = 0
Print IsNull(b) // result True
x% = Null
Print IsNull(x%) // result False
Print
Print "Press any key to close"
KeyGet x%
CloseW 1
```

Remarks

Null is commonly used in database applications to indicate unknown or missing data. Because of the way it is used in databases, **Null** has some unique characteristics:

- Expressions involving **Null** always result in **Null**. Thus, **Null** is said to "propagate" through expressions; if any part of the expression evaluates to **Null**, the entire expression evaluates to **Null**.

- Passing **Null**, a Variant containing **Null**, or an expression that evaluates to **Null** as an argument to most functions causes the function to return **Null**.

- **Null** values propagate through intrinsic functions that return Variant data types.

Null should not be confused with the **Empty** value which is used to indicate an uninitialized Variant variable or **Missing** which is used to indicate an optional **Variant** or **String** parameter was not passed. Furthermore, a value of 0 (zero) or a zero-length string in a **Variant** is not the same as **Null**.

See Also

Nothing, IsNothing, Empty, IsEmpty, Missing, IsMissing

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

Variant Error Type

Syntax

Boolean = **IsError**(*Variant*)

Description

Error values are created in Virtual Basic by converting real numbers to error values using the **CVErr** function. The **IsError** function is then used to determine if a numeric expression represents an error. **IsError** returns **True** if the *expression* argument indicates an error; otherwise, it returns **False**.

GFA-BASIC 32 does not support CVErr, and thus a variant can contain an error value (VT_ERROR) only when an automation object returns such a value (theoretically) or when an optional variant parameter is missing.

Therefore, to make full use of this keyword, you can create a custom **CVErr** functions and use it as follows:

```
Print IsError(test("String"))
Print "Error Number: "; CVErrRead(test("String"))
Print IsError(test(6))
Print "Error Number: "; CVErrRead(test(6))
Print IsError(test())
Print "Error Number: $"; Hex(CVErrRead(test()),
  8)    // The Error Number of the Missing value

FunctionVar test(Optional param1)
  If IsMissing(param1) : test = Missing
     // If no parameter passed, return Missing
  Else If Not IsNumeric(param1) : test =
    CVErr(2001) // Sets a custom error number 2001
    to the return value
  Else : test = param1
  EndIf
EndFunction

Function CVErr(errno%)
  Local vnt As Variant
  DPoke V:vnt, 10
      // Sets the VarType to 10 (VT_ERROR)
  LPoke V:vnt + 8, errno%
      // Sets the value to the error number
  Return vnt
EndFunction

Function CVErrRead(errvnt As Variant)
  If VarType(errvnt) <> 10 Then Return 0
       // If errvnt not an Error then return 0
  Return LPeek(V:errvnt + 8)
       // Else read and return the error number
EndFunction
```

See Also

IsDate, IsEmpty, IsMissing, IsNull, IsNumeric, IsObject

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

Accepted Variable Types for Variants

GFA-BASIC32 supports a large number of variable types that it is possible to enclose in a Variant, but by no means all. The supported types, which largely mirror those supported as standalone variable types (with the notable exception of Large Integers and Fixed Strings), are listed below:

VarType	Value	Description	TypeName
basEmpty	0	Uninitialized	"Empty"
basNull	1	Null (no valid data)	"Null"
basShort	2	Short, 16 Bit Integer (&)	"Short"
basLong **basInt**	3	Integer, Long, 32 Bit-Integer (%)	"Long"
basSingle	4	Single precision floating-point number, 4 Byte (!)	"Single"
basDouble	5	Double precision floating-point number, 8 Byte (#)	"Double"
basCurrency	6	Currency (@)	"Currency"
basDate	7	Date	"Date"

basVString	8	String in Variant	
basObject	9	OLE Automation object	"Object"
//basError	10	Error	"Error"
basBoolean	11	Boolean value (0 or -1)	"Boolean"
basVariant	12	Variant (used only with arrays of Variants)	"Variant"
//basDataObject	13	Non-OLE Automation object	
basByte	17	Byte	"Byte"
//basArray	8192	Array	

The **Missing** value does not have its own variable type but is stored as an **Error** with value $80020004 as shown by the code below:

```
Local vnt As Variant = Missing
Print VarType(vnt), TypeName(vnt)
Print "Error Code for Missing: "; Hex(LPeek(V:vnt
  + 8))
```

As can be seen from the above example, the **VarType** function can be used to return an integer value indicating the type of a variable or the subtype of the variant variable, while the **TypeName** function returns a **String** indicating the type.

The **VarType** function never returns the value for Array by itself. It is always added to some other value to indicate an array of a particular type. The value for Variant is only returned when it has been added to the value for Array to indicate that the argument to the VarType function is an array. For example, the value returned for an array of integers is calculated as 2 + 8192, or 8194. Similarly, **Typename** returns the name of the variable type followed by a pair of brackets '()' to indicate that it is an array, as shown by this example:

```
Local vnt As Variant = Array(1, 2, 3) As Int16
Print VarType(vnt), TypeName(vnt)       // Prints
 8194 and Short()
```

Variant Main Page

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

Programming GFA-BASIC 32 Editor Extensions

The GFA Editor Extensions are a standardized interface for the source code editor of GFA-BASIC 32 and extend the editor with a set of special commands. The interface consists of commands and functions to manipulate source code text, file I/O, and many IDE issues. Meaningful extensions could be an auto save function, automatic minimizing the editor window before running the program and restoring after the program ends, invoking the help file, merging a large number of internal files (:Files) again and again, and inserting code snippets.

Editor extensions are programmed in GFA BASIC 32, so you use the development environment also for creating GLL extensions. Consider however, that contrary to other GFA BASIC 32 projects, the Editor Extensions cannot be run (F5) from inside the IDE. Instead, an editor extension must be compiled and installed before it can be used.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Editor Extension Reference

GFA-BASIC 32 enables you to automate development tasks in the GFA-BASIC 32 development environment. To access the IDE about 130 commands and functions are available. This section describes each command and function

Register Functions

Debugging

Variables and Types

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Compiling and Installing

When a project is marked as a GLL project (.GLL is the extension of a compiled GFA Editor Extension file) the Compile-Dialog box displays an additional tab, called 'Create editor gll'. The tab provides an easy way to initialize the name of the GLL.

The Create Exe Dialog Box

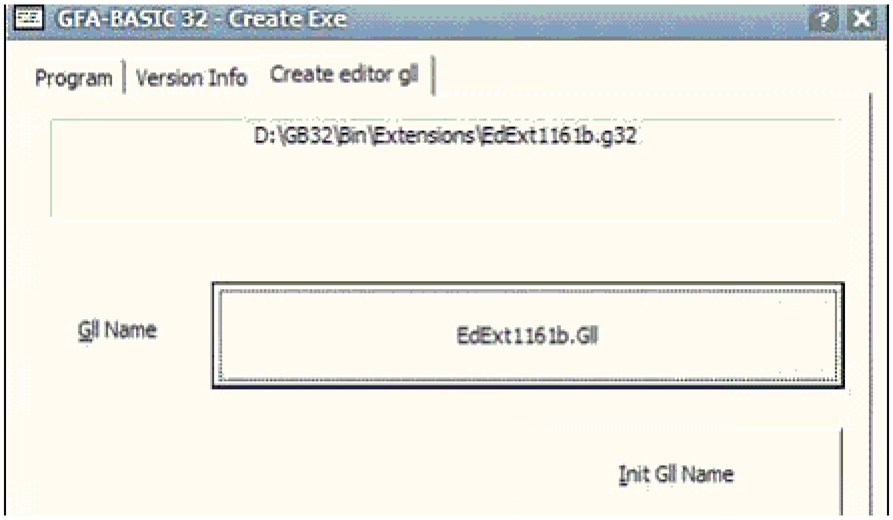
Installing the GLL

Assigning the Keys

Testing a GLL

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

The Create Exe dialog box



D:\GB32\Bin\Extensions\EdExt1161b.g32

Gll Name: EdExt1161b.Gll

Initially the big button next to 'Gll Name' is empty. When you click the 'Init Gll Name' button, the big button above it, is initialized with the filename of the g32 file. In addition the extension is changed to "Gll'. When you want the name to be quite different from the suggested name, click the big button. You can then specify a custom filename for the compiled GLL.

You can still fill in the file version info in the 'Version Info' tab (don't forget to press the small button with + to increment the file version number once a day).

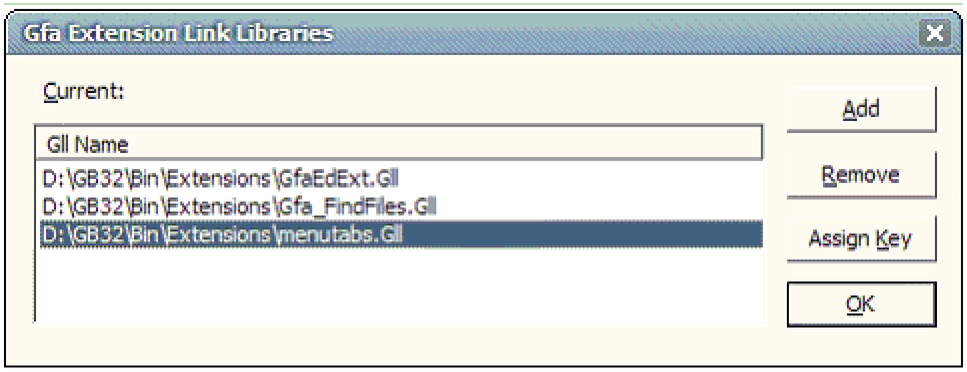
The Program tab can be used as well. The project can still be compiled to an EXE, but all **Gfa_** statements are ignored. It is possible to create a project that combines the functionality of a program and a GLL. For instance, a program might contain the logic to search for text in files. The project might then contain an interface to start the search from within a normal program. Additionally, the program may contain a GLL interface (keyboard shortcut or menu event) that starts the search logic as an editor extension.

Back to the compile process. After providing the file version info and initializing the Gll filename click OK to start compiling. In addition, the information provided in the dialog box is saved in the project file. Note that because the project is extended with the compile information it must be saved again, it has become 'dirty' again (see Gfa_Dirty).

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Installing the GLL

Before the editor extension can be used, it must be installed using the Extension Manager (in German "Gll Laden+Config") which you can find in the Extra submenu.



D:\GB32\Bin\Extensions\GfaEdExt.Gll
D:\GB32\Bin\Extensions\Gfa_FindFiles.Gll
D:\GB32\Bin\Extensions\menutabs.Gll

The Extension Manager is a dialog box named "Gfa Extension Link Libraries" and shows the currently loaded GLLs. They are displayed in the order they are loaded.

To install a new GLL select the Add (or in German "Hinzufügen") button and select the required GLL from the File Open dialogbox. As soon as the GLL is added, it is loaded into memory and, when available, the Gfa_Init sub is executed.

At this point, the installation is not yet complete, though. The keyboard shortcuts that the editor extension wants to use, must be activated. This means that the keyboard shortcut the extension wants to use must be assigned to that sub. By naming the sub Gfa_App_2 we want to execute the sub App + 2 is pressed, but what if this shortcut is already in use by a previously loaded GLL? In that case, we must assign another keyboard shortcut to the Gfa_App_2 subroutine.

The editor extension remains active until it is removed. To remove a GLL use the Extension Manager and choose Remove ("Entfernen"). Before the GLL is unloaded, the Gfa_Exit sub is executed. This provides the opportunity to cleanup resources the GLL used.

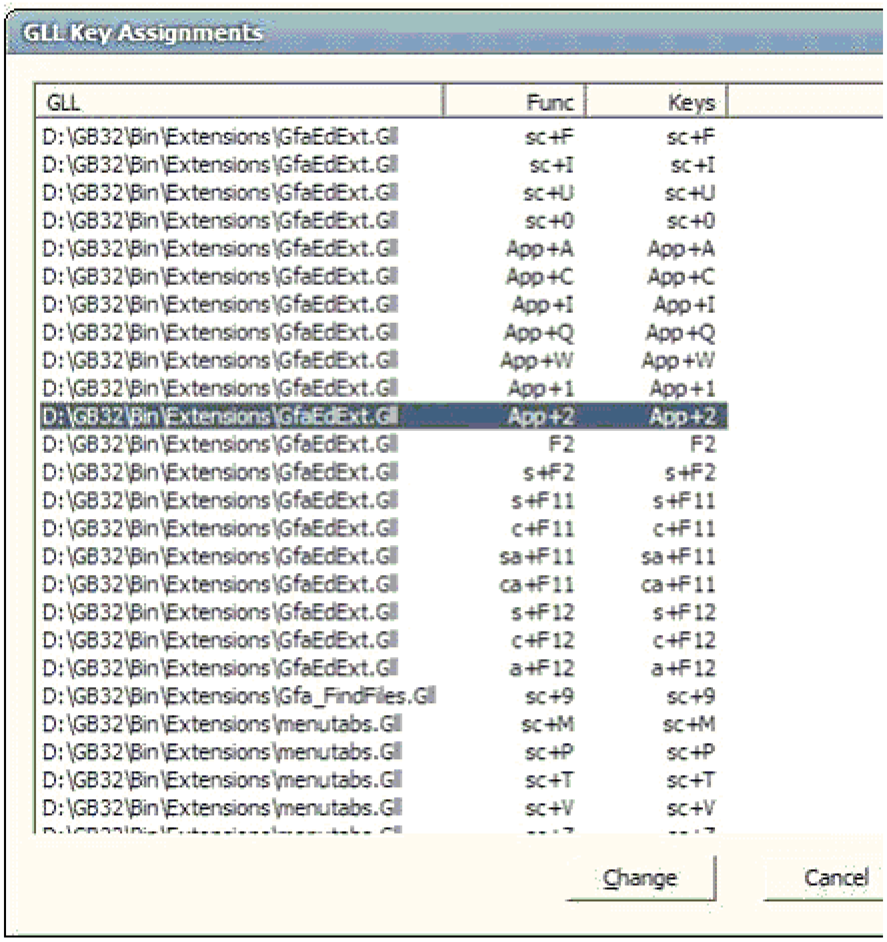
After an editor extension is successfully added to the IDE (installed), it is entered in the registry in HKCU\Software\GFA\Basic section. The next time GFA-BASIC 32 is started all editor extensions that are present in the registry are loaded automatically. They are loaded in the order as they are displayed in the Extension Manager dialog.

i When a GLL behaves badly when GFA-BASIC 32 starts, you might want to remove it from the registry. The GLL keys are named "Gll1", "Gll2", etc.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Assigning the keys

To manage the keyboard shortcuts for all loaded GLLs, click the Assign Key button in the "Gfa Extension Link Libraries" dialog box. Now you'll see the "GLL Key Assignments" dialog box.


GLL Key Assignments

GLL	Func	Keys
D:\GB32\Bin\Extensions\GfaEdExt.Gll	sc+F	sc+F
D:\GB32\Bin\Extensions\GfaEdExt.Gll	sc+I	sc+I
D:\GB32\Bin\Extensions\GfaEdExt.Gll	sc+U	sc+U
D:\GB32\Bin\Extensions\GfaEdExt.Gll	sc+0	sc+0
D:\GB32\Bin\Extensions\GfaEdExt.Gll	App+A	App+A
D:\GB32\Bin\Extensions\GfaEdExt.Gll	App+C	App+C
D:\GB32\Bin\Extensions\GfaEdExt.Gll	App+I	App+I
D:\GB32\Bin\Extensions\GfaEdExt.Gll	App+Q	App+Q
D:\GB32\Bin\Extensions\GfaEdExt.Gll	App+W	App+W
D:\GB32\Bin\Extensions\GfaEdExt.Gll	App+1	App+1
D:\GB32\Bin\Extensions\GfaEdExt.Gll	App+2	App+2
D:\GB32\Bin\Extensions\GfaEdExt.Gll	F2	F2
D:\GB32\Bin\Extensions\GfaEdExt.Gll	s+F2	s+F2
D:\GB32\Bin\Extensions\GfaEdExt.Gll	s+F11	s+F11
D:\GB32\Bin\Extensions\GfaEdExt.Gll	c+F11	c+F11
D:\GB32\Bin\Extensions\GfaEdExt.Gll	sa+F11	sa+F11
D:\GB32\Bin\Extensions\GfaEdExt.Gll	ca+F11	ca+F11
D:\GB32\Bin\Extensions\GfaEdExt.Gll	s+F12	s+F12
D:\GB32\Bin\Extensions\GfaEdExt.Gll	c+F12	c+F12
D:\GB32\Bin\Extensions\GfaEdExt.Gll	a+F12	a+F12
D:\GB32\Bin\Extensions\Gfa_FindFiles.Gll	sc+9	sc+9
D:\GB32\Bin\Extensions\menutabs.Gll	sc+M	sc+M
D:\GB32\Bin\Extensions\menutabs.Gll	sc+P	sc+P
D:\GB32\Bin\Extensions\menutabs.Gll	sc+T	sc+T
D:\GB32\Bin\Extensions\menutabs.Gll	sc+V	sc+V
[illegible]	[illegible]	[illegible]

The dialog box lists all keyboard shortcuts subs in the currently loaded GLLs.

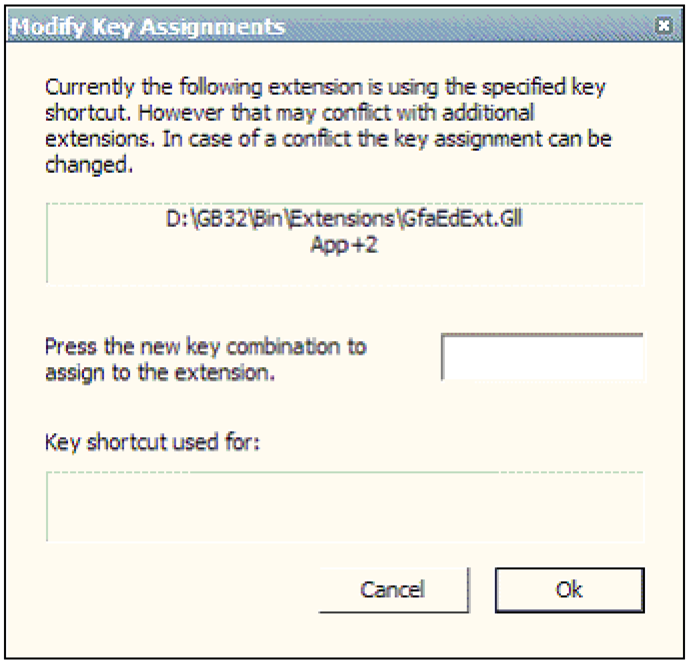
The first column specifies the editor extension and the second column the Sub (Func) that is contained in that extension.

For instance, when the loaded GLL extension "D:\GB32\Bin\Extensions\GfaEdExt.Gll" contains a sub **Gfa_Ex**_F, then the column 'Func' specifies "sc+F". If it contains a sub **Gfa_CAF11** then the Func column shows ca+F11, etc.

The third column 'Keys' specifies which keyboard shortcut is actually assigned to the subroutine Func. Initially, the value for 'Keys' is empty. By double clicking the entry in the list box or by clicking the command button 'Change' you can specify which keyboard shortcut is to execute the event.

In the picture, the Sub **Gfa_App**_2 uses App + 2 for executing the sub. To override the default settings, select the button Assign Key ("Tastatur") in the Extension Manager dialog box.

To change a key assignment select the GLL sub (Func) you want to re-assign and select Change ("Tastenbelegung ändern"). The following dialog box is displayed.


Modify Key Assignments

Currently the following extension is using the specified key shortcut. However that may conflict with additional extensions. In case of a conflict the key assignment can be changed.

D:\GB32\Bin\Extensions\GfaEdExt.Gll
App+2

Press the new key combination to assign to the extension.

Key shortcut used for:

In the Modify Key Assignments you can assign a new key combination. Press the key combination you want to use for the selected function and see if it is used. When the key combination is already in use, the GLL it is assigned to is displayed below. Choose a different keyboard shortcut and close the dialog box.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Testing a GLL

During the development of an editor extension, you have two options to load the newly compiled GLL. First, you must make sure it is entered in the Extension Manager list. Since this operation loads the GLL as well, you can test immediately. All other times you could repeat this process but remove the GLL first.

Unfortunately, replacing a GLL using the Extension Manager requires quite some actions to perform the task. Another way to load a GLL to test it is by restarting the IDE after compiling a GLL. This takes two shortcuts: Alt-F4 to quit GFA-BASIC 32 and a Windows shortcut key to start the GFA-BASIC 32 IDE.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Restrictions and Features

Not all GFA-BASIC 32 commands and functions may be used in an editor extension. A GLL differs in quite some ways from a normal GFA-BASIC 32 application. GFA-BASIC 32 applications are constructed around the OLE wrappers for windows, forms and controls. This means that maintaining all GUI items is performed through COM calls. The forms and windows are COM containers for the ActiveX controls provided in the GfaWin23.OCX. The GFA-BASIC 32 application is able to communicate between the COM items through the Sleep command, which is used in any normal program.

The editor extension is an external compiled GFA-BASIC 32 program in a special format. The editor extensions are programmed in GFA-BASIC 32 and make use of many library functions provided by GFA-BASIC 32, but a GLL is not a COM plug-in, it is not COM based. After loading a GLL plug-in it will become part of the IDE, its functions are called from inside the IDE, which is a regular WINAPI program and not a COM program. The IDE has no knowledge what so ever about COM containers and OCXs. As a consequence a GLL cannot use Form, OpenW, Ocx, Sleep, etc. The general rule is: don't use GFA-BASIC 32 GUI commands and don't use GFA-BASIC 32 specific message loops, not even GetEvent in a GLL.

Editor Extensions do not have a data segment, because they are nothing more then a piece of compiled code that is recognized by the IDE. Also, a GLL is not a DLL, it cannot contain data and cannot contain resources. Therefore, **Read, Data, Restore, and LabelAddr()** are not allowed. Since a GLL has no data section, inline resource files (:Files) are not allowed as well. (There is a workaround to include data in a GLL. By encoding binary data in a MimeEncode$ format, the data can be assigned to a string variable and later decoded.)

Allowed are mostly all other GFA-BASIC 32 functions. You can open files, use the non GUI COM objects App, Screen, Debug, Err, Printer, Collection, and DisAsm. All Windows API functions may be used, there are hardly any limitations.

You can display (test) results using **Debug.Print**, Trace, Assert, and MsgBox, or by inserting text into program text, as well as by changing the status bar text Gfa_StatusText**=**.

Simple input for a GLL can take place with Gfa_KeyGet, InputBox**,** Prompt**, or** Popup.

This manual contains many examples that clarify the usage of GFA-BASIC 32 statements in GLLs.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

The Structure of an Editor Extension

GFA Editor Extensions have no main program and cannot be executed (F5). Initializations, otherwise made in the main part of a GFA BASIC 32 program, must take place in the Sub with the name Gfa_Init. This sub is executed automatically while loading the editor Extension of the GFA BASIC 32. This sub is also the place for Gfa_AddMenu, in order to add entries to the Extra submenu.

```
Global Enum LangEng = 0, LangGer         ' values
 CurrentLanguage

Sub Gfa_Init
  Global CurrentLanguage As Int =
    Gfa_IntSetting("Language")
  If CurrentLanguage = LangEng  // English
    IdxMerge = Gfa_AddMenu("Insert file ...",
      Gfa_MenuMerge)
    Gfa_MenuDesc(IdxMerge) = "Inserts the contents
      " _"
  Else
    IdxMerge = Gfa_AddMenu("Merge Datei ...",
      Gfa_MenuMerge)
    Gfa_MenuDesc(IdxMerge) = "Merge Datei .."
  EndIf
  '
  ' Create a font resource …
  Global Handle hMyFont = CreateMyFont("Arial")
End Sub
```

The menu entries are removed automatically when the GLL is unloaded. Any resource allocation can be released in the

Gfa_Exit sub, which is automatically executed when the GLL is unloaded from memory.

```
Sub Gfa_Exit
  ~DeleteObject(hMyFont)
EndSub
```

Using Keyboard Shortcuts

Using The Extra Menu

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Using Keyboard Shortcuts

To call an editor extension function you must create event subroutines with names that identify the keyboard shortcuts they must respond to. These keyboard subs have the fixed names **Gfa_Ex**_?, **Gfa_App**_? or **Gfa_App**_S?, where ? is a placeholder for one of the characters A-Z and the numbers 0-9. Thus, when you want to create an extension procedure that is invoked after pressing the combination Shift+Ctrl+X, the subroutine should be named **Gfa_Ex**_X.

```
Sub Gfa_Ex_X     ' Shift+Ctrl+X key event
  ' Todo: your extension code
EndSub
```

Combinations with the application key are allowed as well. For App+X the sub **Gfa_App**_X is called. The App key is the Windows application key (Application key), which sits next to the right Windows Start key. Often this key is used to display a context menu, which might also be a good purpose for an editor extension.

Example: Insert Date and Time

```
Sub Gfa_App_D    ' App+D - popup to insert Date &
  Time
  Dim i% = PopUp(" Date| Time| DateTime")
  Gfa_Insert Choose(i% + 1, Date$, Time$, Now$)
EndSub
```

There are also some function keys available for shortcut assignment: F2, F8, F9, and in shifted states for F11 and F12.

Shift keys	**Subs**
None	Gfa_F2, Gfa_F8, Gfa_F9
Shift	Gfa_SF2, Gfa_SF8, Gfa_SF9, Gfa_SF11, Gfa_SF12
Ctrl	Gfa_CF2, Gfa_CF8, Gfa_CF9, Gfa_CF11, Gfa_CF12
Shift + Ctrl	Gfa_SCF2, Gfa_SCF8, Gfa_SCF9, Gfa_SCF11, Gfa_SCF12
Alt	Gfa_AF2, Gfa_AF8, Gfa_AF9, Gfa_AF11, Gfa_AF12
Shift + Alt	Gfa_SAF2, Gfa_SAF8, Gfa_SAF9, Gfa_SAF11, Gfa_SAF12
Ctrl + Alt	Gfa_CAF2, Gfa_CAF8, Gfa_CAF9, Gfa_CAF11, Gfa_CAF12
Shift + Ctrl + Alt	Gfa_SCAF2, Gfa_SCAF8, Gfa_SCAF9, Gfa_SCAF11, Gfa_SCAF12

Note: S = Shift, C = Ctrl, A = Alt, SCA = Shift + Ctrl + Alt

Example:

```
Sub Gfa_CF2                ' Ctrl+F2 - New
  Gfa_New
End Sub
```

Using The Extra Menu

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Using The Extra Menu

Another possibility to invoke an editor extension function is to respond to a menu event from a previously added menu entry in the Extra submenu. This menu contains no entries by default. The only possibility to add entries is through the use of the Extensions.

Use the instruction **Gfa_AddMenu** to inserted new menu entries into the extra menu. **Gfa_AddMenu** expects the name of a Sub as the second parameter associated with the menu option. When the menu entry is chosen this Sub is executed. For instance:

```
Sub Gfa_MenuMerge(Idx%)
  ' handle menu event
EndSub
```

Event Subs

The third interface to the editor Extensions are the event-controlled Subs. Comparably with the event Subs of Ocx objects (see object manual), it concerns subroutines with a descriptive name, when occurring a certain event to be called automatically and executed.

For example when a procedure with the name Sub **Gfa_Minute** is present in the editor Extension, this procedure is called and executed every minute (the editor is interrupted as long as it takes to carry out the steps in timer event sub).

The following event subs are implemented:

Gfa_Init - Occurs when a GLL is loaded.

Gfa_Exit - Occurs when a GLL is unloaded.

Gfa_OnRun - Occurs when a g32 project is run.

Gfa_OnEnd - Occurs when a run program ends.

Gfa_Second - Occurs every second.

Gfa_Minute - Occurs every minute.

Gfa_OnDropInl - Occurs when files are dropped on the :Files window.

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Using Dialogs in a GLL

Although the GFA Editor Extensions are programmed using GFA-BASIC 32 statements and functions, they can not use windows, forms, ocxs and normal message processing commands like, for instance, **Sleep**. In fact, the editor extensions do not allow any of the usual GFA-BASIC window functions. There is one exception however. You can use the **Dialog**/**EndDialog** structure to create modeless dialog boxes as in GFA-BASIC 16 bit. These commands are implemented by invoking functions inside the IDE, not by compiling them to normal GFA-BASIC 32 application instructions. The modeless dialog box is therefore part of the IDE and its messages are retrieved in the IDE's main message loop.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Ownerless and modeless

The editor extension dialog boxes are both ownerless and modeless. Two relationships that can exist between windows are the owner-owned relationship and the parent-child relationship. The owner-owned relationship determines which other windows are automatically destroyed when a window is destroyed. When window A is destroyed, Windows automatically destroys all of the windows owned by A. The parent-child relationship determines where a window can be drawn on the screen. A child window (that is, a window with a parent) is confined to its parent window's client area.

The default window style of an editor extension dialog is WS_POPUP | WS_CAPTION. The dialog is created using a call to *CreateWindowEx* with *hWndParent* set to **Null**. The dialog box is owned by the desktop and not by the IDE. When the dialog box is being owned it would places several constraints on the dialog box.

- An owned window is always above its owner in the Z order.

- The system automatically destroys an owned window when its owner is destroyed.

- An owned window is hidden when its owner is minimized.

By making the dialog box ownerless, these constraints are now removed; there is no relation between the IDE main window and the editor extension dialog boxes. To maintain its visual view the extension must process window messages and handle the appropriate message by its self.

A modeless dialog box does not disable its parent or owner when it is displayed as modal dialog box would. A modal dialog box must be closed before the parent is accessible again. From the fact that the dialog isn't owned it is clear that the dialog box is modeless.

By disabling the IDE window (Gfa_hWnd**)** you can simulate a modal dialog box. However, when another application is activated and the dialog box is set back in the Z-order it is not very easy to make it visible again. You cannot click on the IDE because it is disabled and the dialog box is not visible in the taskbar. You must minimize other applications before you can access the dialog box again. There is no easy solution to this problem.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

The Dialog Statements

The following editor extension commands are syntactically the same as the GFA-BASIC 16 bit commands, but operate on a different level: inside the IDE.

Dialog #id, x%,y%,w%,h% [,title$ [,style% [,fontheight%,fontname$]]]

EndDialog ShowDialog #id
CloseDialog #id

By default the coordinates specify the number of pixels, an implicit DlgBase **Pixel**. By using **DlgBase Unit** the interpretation of the dialog box and control coordinates is changed.

The default style is WS_POPUP. When the title argument is specified the WS_CAPTION is set as well. In most cases only WS_SYSMENU is provided as the argument for style.

The default font is the font obtained using GetStockObject(DEFAULT_ GUI_FONT), which will suffice in most circumstances.

ShowDialog is implemented as *ShowWindow*(**Dlg**(id),SW_SHOW).

The GWL_USERDATA index with the *SetWindowLong* API function should not be used. GFA-BASIC 32 reserves the value at this index to store a pointer to a dialog info block for the dialog box.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Creating controls

Try to avoid the general **Control** statement to create a child window, these controls use the system font, rather than the DEFAULT_ GUI_FONT.

Instead use GFA-BASIC 32 control statements for the different kinds of standard and common controls. For instance, to create a simple left justified static text control:

LText text$, ID%, x%, y%, width%, height% [,style%]

All control statements use the same syntax:

CtrlName *text$, ID%, x%, y%, width%, height% [,style%]*

CtrlName	Name of the GFA-BASIC 32 control statement.
text$	Specifies text that is displayed with the control. The text is positioned within the control's specified dimensions or adjacent to the control.
ID%	Specifies the control identifier. This value must be an integer in the range 0 through 65,535 or a simple arithmetic expression that evaluates to a value in that range.
x%, y%	Specifies the x- and y-coordinate of the left top side of the control relative to the left top side of the dialog box. The coordinate is in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width%	Specifies the width of the control.
height%	Specifies the height of the control.

style%	Specifies the control styles. Use the bitwise OR (|) operator to combine styles.

Standard controls:
LText, RText, CText, Icon,
PushButton, DefPushButton, CheckBox, AutoCheckBox,
RadioButton, AutoRadioButton,
ListBox, ComboBox,
EditText, Scrollbar.

Common Controls:
AnimateCtrl,
TabCtrl,
HeaderCtrl, ListViewCtrl, TreeViewCtrl,
ProgressCtrl, TrackBarCtrl,
StatusCtrl, ToolBarCtrl,
UpDownCtrl.

Other: RichEditCtrl.

Note - There is no **Static** control command, Static is used to declare static local variables. Use the general Control statement instead.

Note - GFA-BASIC 32 also provides keywords like ProgressBar, Toolbar, Header, etc. These keywords are not statements to create controls, but they are OCX *types*. As such these keywords are used to declare variables or to create OCX controls. For instance:

```
Dim pb As ProgressBar ' declare a variable pb
Ocx ProgressBar pb1    ' create OCX & declare
 global variable pb1
```

These OCX types are not allowed in a GLL.

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Other Window Commands in GLLs

Everything that has to do with OpenW, LoadForm, OCX, Sleep, GetEvent, MENU(), etc is not allowed. However, for easy access, the following GFA-BASIC statements and functions are implemented to be used in Gfa Editor Extensions.

ExtensionImplementation

Dlg(id) - Obtains the window handle of the dialog with number id (0 .. 31)

DlgItem(id,idc) - Obtains the window handle of the child control idc in dialogbox id.

c$ = _Win$(h) - Returns a string with the window text of the window with handle h.

_Win$(h) = c$ - Sets the window text of window h with the contents of string c$.

MoveW id, x, y - *SetWindowPos*(Dlg(id),0,x,y,0,0,SWP_NOZORDER | SWP_NOSIZE)

SizeW id, w, h - *SetWindowPos*(Dlg(id),0,0,0,w,h,SWP_NOZORDER | SWP_NOMOVE)

CloseW id - *DestroyWindow*(Dlg(id))

ClearW id - *InvalidateRect*(Dlg(id),0,1) + UpdateWindow(Dlg(id))

ShowW id,swf - *ShowWindow*(Dlg(id), swf)

EnableW id - *EnableWindow*(Dlg(id), 1)

DisableW id - *EnableWindow*(Dlg(id), 0)

Enabled?(id) - *IsWindowEnabled*(Dlg(id))

SetCheck id, n, f - *SendMessage*(Dlg(id,n), BM_SETCHECK,f,0)

Check?(id, n) - f = *SendMessage(* Dlg(id,n), BM_GETCHECK,0,0)

Zoomed?(id) - *IsZoomed*(Dlg(id))

Visible?(id) - *IsWindowVisible*(Dlg(id))

Iconic?(id) - *IsIconic*(Dlg(id))

Except for Dialog**#**, ShowDialog, and CloseDialog, all functions take either a dialog number (between 0 and 31) or a window handle.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Message Handling using Gfa_CB

The message handling for the dialog does not take place using PeekEvent, GetEvent, DoEvents, or Sleep. Instead message processing is part of the main message loop of the GFA-BASIC IDE program. Whenever a message for GFA Editor Extension dialog box arrives it is dispatched to the **Gfa_CB** sub which should handle your dialog box message.

The syntax for the dialog callback sub is:

Sub **Gfa_CB**(*hDlg%, Msg%, wParam%, lParam%, RetVal%, ValidRet?*)

The *hDlg*% parameter is the window (dialog) to which the message is sent. The *Msg*% parameter is the message number, which is usually a constant such as WM_COMMAND or WM_SIZE. The *wParam%* and *lParam%* parameters differ for each message, as does the return value; you must look up the specific message to see what they mean. Often, wParam or the return value is ignored, but not always.

The **Gfa_CB** has two additional parameters (ByRef) that allow you to return a value to default window procedure for the dialog box. For instance, when you handled a certain message you can set the *ValidRet*? variable to True and provide a return value by setting the *RetVal*% variable. What value *RetVal* must have is defined in the Windows API SDK. It often says something like: "If you handle this message return zero (or..)".

There can be only one **Gfa_CB** sub per editor extension. So, inside the **Gfa_CB** sub you must determine which dialog # the message is for. Note that a dialog box is defined using an ID number (0..31) and that the dialog is accessed using its ID number.

To obtain the target dialog box number you must obtain the ID number from the *hDlg%* parameter, which specifies the windows handle for the dialog box. The window handles for the dialog boxes are retrieved using the Dlg() function. **Dlg**(id) expects a number between 0 and 31 and returns the window handle of that dialog box. The target dialog box ID is determined by iterating over the dialog box ID values, 0 to 31, until hDlg% equals **Dlg**(id).

The following example illustrates the message handling mechanism in a GFA Editor Extension for three dialogs:

```
Sub Gfa_CB(hDlg%, Msg%, wParam%, lParam%, RetVal%,
  ValidRet?)
  If hDlg% = Dlg(1)
    Handle_Dlg1(hDlg%, Msg%, wParam%, lParam%,
      RetVal%, ValidRet?)
  Else If hDlg% = Dlg(2)
    Handle_Dlg2(hDlg%, Msg%, wParam%, lParam%,
      RetVal%, ValidRet?)
  Else If hDlg% = Dlg(3)
    Handle_Dlg3(hDlg%, Msg%, wParam%, lParam%,
      RetVal%, ValidRet?)
  EndIf
EndSub

Sub Handle_Dlg1(hDlg%, Msg%, wParam%, lParam%,
  RetVal%, ValidRet?)
  // Code
EndSub
```

```
Sub Handle_Dlg2(hDlg%, Msg%, wParam%, lParam%,
 RetVal%, ValidRet?)
 // Code
EndSub

Sub Handle_Dlg3(hDlg%, Msg%, wParam%, lParam%,
 RetVal%, ValidRet?)
 // Code
EndSub
```

Once a message is arrived in the callback subroutine, there are two alternatives in processing possible.

1. Create a large switch case branch based on the Msg% variable and handle the message directly (preferred).

2. Store the parameters in global variables and process the message in Gfa_Second. This is kind of the same as messages were handled in GFA-BASIC for Windows 3.1 using MENU(). (This method is mentioned by GFA, but it is not recommended.)

{Created by Sjouke Hamstra; Last updated: 07/07/2015 by James Gaite}

Example: Using a Dialog

In the following example a dialog box identified with number # 1 is displayed when the editor extension keyboard shortcut Shift+Ctrl+9 is pressed. Per editor extension 31 dialogbox can be displayed simultaneously. Their id numbers range from 1 to 31.

```
Sub Gfa_Ex_9
  Dialog # 1, 10, 10, 100, 170, "TestDlg",
    WS_SYSMENU
    PushButton "but &A", 100, 10, 10, 50, 20
    PushButton "but &B", 101, 10, 32, 50, 20
    PushButton "but &C", 102, 10, 54, 50, 20
    DefPushButton "Ok", IDOK, 10, 76, 50, 20
    PushButton "Cancel", IDCANCEL, 10, 98, 50, 20
    EditText "", 200, 10, 120, 50, 22
  EndDialog
  ShowDialog # 1
  ~SetFocus(Dlg(1, IDOK))
EndSub

Sub Gfa_CB(h%, m%, w%, l%, r%, f?)
  If h% = Dlg(1)
    Switch m
    Case WM_COMMAND
      Switch w
      Case 100 : MsgBox "Button A pressed"#10 &
        _Win$(Dlg(1, 200))
      Case 101 : cmdB_Click
      Case 102 : MsgBox "Button C pressed"
      Case IDOK : MsgBox "Ok pressed" #10 &
        _Win$(Dlg(1, 200)) : CloseDialog # 1
      Case IDCANCEL : MsgBox "Cancel" : CloseDialog
        # 1
      EndSwitch
    Case WM_CLOSE
      MsgBox "Cancel - Close"
    EndSwitch
  EndIf
EndSub

Sub cmdB_Click
  MsgBox "Button B pressed" #10 & _Win$(Dlg(1,
    200))
EndSub
```

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

Problem with menu events

When a dialog box is created and showed as a result of processing a menu event, the dialog box is overlapped by the GFA-BASIC 32 IDE immediately. The IDE is brought to foreground after showing the dialog. This is a direct result from the internal WM_COMMAND handling of the IDE: it simply set the focus to the editor after handling the WM_COMMAND message, e.g. the menu event. It doesn't check for a visible dialog box. This problem only occurs after a menu event, not with a keyboard event.

One workaround could be by posting the keyboard shortcut that creates the dialog.

```
Sub menuDialog(i%)
  SendKeys "^+9"          'calls Gfa_Ex_9 eventually
EndSub
```

This way the keyboard shortcut is placed in the message queue and will not be retrieved before the menu event is handled completely and the main message loop is re-entered.

Another solution is to relocate the ShowDialog command to the Gfa_CB procedure. Then, rather than invoke **ShowDialog** and *SetFocus* post a WM_USER message to display dialog. The advantage of this approach is that the initialization of the dialog box can be done in the Gfa_CB as well. The entire handling of the dialog box can be combined in one procedure.

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Error Handling

Internally, the call to **Gfa_CB** is embedded in a **Try/Catch** structure. An error inside the **Gfa_CB** is therefore trapped and logged to the Debug Output window. Nevertheless, to process error conditions properly, the **GFA_CB** should have its own **Try/Catch** structure.

See Also

Try

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Miscellaneous Examples

Below a list with links to some GLL examples.

AutoSave

Change Case

Convert Characters

Using Eval()

Insert Snippet Code

Add a Resource

Jump to subroutine

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Arrays Keyword Summary

Action	Keywords	GB	VB 6
Verify an array	IsArray	v	v
Create an array in a Variant	Array	v	v
Change default lower limit	Option Base	v	v
Declare and initialize an array	Dim, Private, Public, ReDim, Static	v	v
Find the limits of an array	LBound, Ubound	v	v
The number of elements	Dim?	v	
The number of indices	IndexCount	v	
Reinitialize an array	Erase, ReDim	v	v
Insert and delete elements	Insert, Delete	v	
Array address and size	ArrayAddr, ArraySize	v	
Array initialize	ArrayFill	v	
Sort array	QSort	v	
Write/Read string array	Recall, Store	v	

Note **Private** is a synonym to **Local** and **Public** is the same as **Global**.

See Also

GFABasic32 Language Reference

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Bits and Byte Operators and Keywords

Action	Keywords	GB	VB6
Change and test bits of 32-bits integers	Bchg, Bclr, Bset, Btst	v	
Change and test bits of 64-bit integers	Bchg8, Bclr8, Bset8, Btst8	v	
Shift bits (32-bits)	<<, >> Shl, Shr, Sar	v	
Rotate bits (32-bits)	Rol, Ror	v	
Shift bits (64-bits)	Shl8, Shr8, Sar8	v	
Rotate bits (64-bits)	Rol8, Ror8	v	
Swap bytes	Bswap	v	
Exchange bytes	_Swab, _Swab8, _SwabL	v	
Swap bits	Mirror, Mirror%, Mirror&, Mirror|, Mirror8	v	
Extract high and low bytes and words	HiByte, HiCard, HiWord, LoByte, LoCard,LoWord, LoLarge,HiLarge	v	
Extract Card and	Card,Byte	v	

Byte type		
Sign extend	Word, Short, Ushort, Uword	v
Create integers (16-bit, 24-bit, 32-bit, and 64-bit)	MakeL2H, MakeL2L, MakeL3H, MakeL3L, MakeL4H, MakeL4L, MakeLarge, MakeLargeHiLo, MakeLargeLoHi, MakeLong, MakeLongHiLo, MakeLongLoHi, MakeWord, MakeWordHiLo, MakeWordLoHi, MakeWParam	v
Peek numeric values	Peek, CPeek, CurPeek, DPeek, DblPeek, LPeek, Peek8, SngPeek	v
Poke numeric values	Poke, CPoke, CurPoke, DPoke, DblPoke, LPoke, Poke8, SngPoke	v
Network integer conversions	htonl(), htons(), ntohl(), ntohs()	v

See Also

Bits, Byte, Word, Int, and Large Operators and Keywords

Conversion Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Built-In API Functions

There are about 1000 most often used API functions built-in. Some of these functions have reserved names, names that are already in use by the BASIC language. Those functions are renamed and are available under an alias.

A built-in function always uses the return value, either by using it in an expression or by voiding the value. For instance:

```
Dim p%

p% = CharLower("ABC") // expression

~CharLower("ABC") // ~ void

Void CharLower("ABC") // void
```

The Windows API supports both ANSI and UNICODE functions. The GFA-BASIC 32 built-in functions are the ANSI variant.

Passing strings

Many API functions take the C data type LPSTR (or LPTSTR, LPCSTR, etc) as a parameter. This is a pointer to a null-terminated string. When passing a string data type pass the address of the character data using the **VarPtr** function or the **V:** operator.

```
Dim api$ = "ABC"

~CharLower(V:api$) // address of the string
```

When a string is passed 'by value', as in the first example, the string is first copied to an internal buffer of 1030 bytes. Then the address of the buffer is passed to the API function. This has two disadvantages. First any strings larger than 1029 characters are cut off. Secondly, API functions that return text data to a LPSTR buffer are not received by the program, because they are copied to the temporal buffer.

The built-in functions are not checked for a correct syntax, only for the correct number of parameters. In fact, each parameter of the built-in functions is simply a 32-bit integer. What ever you pass to that integer is the responsibility of the programmer.

Passing user-defined types

A user-defiined type can be passed by address using **V:** or 'by value', without the **V:**. GFA-BASIC 32 always passes the address of the type variable.

Renamed API functions

The following API functions are renamed due to there use as reserved keyword for a BASIC command or function. Note some got two new names.

GetObject in **GetGdiObject** or **apiGetObject**

LoadCursor in **LoadResCursor** or **apiLoadCursor**

lstrcmpi in **_lstrcmpi**

lstrcmp in **_lstrcmp**

Removed API functions

The following 16-bit API functions are not included:

GetBitmapDimension, SetBrushOrg, GetBrushOrg, GetCurrentPosition, PostAppMessage, SetConvertHook, SetConvertParams, ConvertRequest, DefHookProc, GetAspectRatioFilter, GetCurrentTask, GetNumTasks, SetSysModalWindow, and UnlockResource

See Also

Declare

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Collection and Hash Keywords

The **Collection**

Action	**Keywords**	**GB**	**VB6**
Create a Collection object	Collection	v	v
Add an object to a collection	Add	v	v
Remove an object from a collection	Remove	v	v
Reference an item in a collection	Item	v	v

The **Hash**

Action	**Keywords**	**GB**	**VB6**
Create a Hash	Hash	v	
Add an element to a Hash	Hash Add	v	
Remove an element form a Hash	Hash Remove	v	
Erase Hash	Hash Erase	v	
Load/ Save a Hash collection	Hash Input, Hash Load, , Hash Save, Hash Write	v	

Sort a Hash by keyword	Hash Sort	v

See Also

Arrays Keywords

Crypting, Mime encoding, Checksum Keywords

Data Types Keywords

Miscellaneous Keywords

OCX/OLE Keywords

String Manipulation Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Control Flow Keywords

Action	Keywords	GB	VB6
Branch	GoSub...Return, GoTo, On Error, On..GoSub, On..GoTo	v	v
Exit or pause the program	DoEvents, End, Exit, Stop	v	v
Exit or pause the program	Sleep, GetEvent, PeekEvent, Quit	v	
Loop	Do...Loop, For... Next, ForEach... Next, While... Wend, With	v	v
Loop	Repeat	v	
Make decisions	Choose, If... Then...Else, Select, Switch, Iif	v	v
Use procedures	Call, Function, Sub	v	v
Use procedures	Procedure, FunctionVar,	v	
Properties	Property Get, Property Let, Property Set		v
Call a function through a pointer	C, LC, LP, Call, CallX, Ccall, LCCall, LpasCall, StdCall, LstdCall	v	

See Also

Compiler and Debug Keywords

Data Types Keywords

Errors Keywords

Memory Keywords

Miscellaneous Keywords

Operators Keywords

String Manipulation Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Compiler and Debug Keywords

Compiler directives and keywords

Action	Keywords	GB	VB6
Use of variable postfix	$AutoPost	v	
Check array bounds	$ArrayChk	v	
Fast For...Next code	$For	v	
Check OLE errors	$Obj	v	
Insert debug code	$Step	v	
Create exports	$Export	v	
Load compiled library	$Library	v	
Minimum prologue and epilogue code for subroutines	Naked	v	
Auto declare global variables	Auto	v	
Running as EXE?	IsExe	v	
Set a hardware breakpoint	Monitor	v	

Debug Keywords

Action	Keywords	GB	VB6
Debug Object	Debug	v	v
Assert	Assert	v	
Dump call stack	Calltree	v	
Trace lines	Tron, Trace, TraceLnr, TraceReg, SrcCode$, ProcLnr, ProcLineCnt	v	

See Also

Errors Keywords

Miscellaneous Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Conversion Keywords

Action	Keywords	GB	VB6
ANSI value to string	Chr	v	v
	#, Mk1$	v	
Convert a string to a numeric data type.	Cv1, Cv2, Cv3, Cv4, Cv5, Cv6, Cv7, Cv8, Cvd, CvdMbf, Cvi, Cvl, Cvs, CvsMbf, Cvw, CvCur, CvLarge	v	
Convert a data type to a string	Mk1, Mk2, Mk3, Mk4, Mk5, Mk6, Mk7, Mk8, MkCur, Mkd, MkdMbf, Mki, Mkl, MkLarge, Mks, MksMbf, Mkw	v	
String to lowercase or uppercase	Format, Lcase, Ucase	v	v
	Upper, Lower	v	
Date to serial number	DateSerial, DateValue	v	v
Decimal number to other bases	Hex$(), Oct$()	v	v
	Bin$(), Dec$(),	v	
	Base	v	

Number to string	Format, Str	v	v
	sprintf, Using	v	
One data type to another	Cbool, Cbyte, Ccur, Cdate, CDbl, Cint, Clong, CSng, CStr, Cvar, Fix, Int	v	v
	CDec, CVErr		v
Convert between data types and round to zero.	CByteRZ, CIntRZ, CLargeRZ, CLongRZ, CShortRZ	v	
String to ASCII value	Asc	v	v
String to number	Val, CDbl	v	v
	Val?	v	

See Also

Data Types Keywords

Dates and Times Keywords

Math Keywords

Miscellaneous Keywords

Operators Keywords

String Manipulation Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 17/10/2017 by James Gaite}

Crypting, Mime encoding, Checksum Keywords

Action	Keywords	GB	VB6
Crypt data	Crypt	v	
Checksum	Crc16, Crc32	v	
Checksum	CheckSumByte, CheckSumLong, CheckSumShort, CheckXorByte, CheckXorLong, CheckXorShort	v	
Pack data	Pack, UnPack, PackMem, UnPackMem	v	
Mime 64 encoding	MemToMiMe, MiMeToMem, MiMeDecode, MiMeEncode,	v	
Mime UU encoding	MemToUU, UUToMem, UUDecode, UUEncode	v	

See Also

Collection and Hash Keywords

Conversion Keywords

Data Types Keywords

{Created by Sjouke Hamstra; Last updated: 17/10/2017 by James Gaite}

Data Types Keywords

Action	Keywords	GB	VB6
Convert between data types	Cbool, Cbyte, Ccur, Cdate, CDbl, Cint, CLong, CSng, CStr, Cvar,, Fix, Int	v	v
	CDec, CVErr		v
Convert between data types and round to zero.	CByteRZ, CIntRZ, CLargeRZ, CLongRZ, CShortRZ	v	
Set intrinsic data types	Boolean, Byte, Currency, Date, Integer, Long, Object, Single, Double, Variant (default)	v	v
Additional intrinsic data types	Card, Short, Word, Int16, Int, Int32, Int64, Large, Handle, , Hash	v	
Verify data types	IsArray, IsDate, IsEmpty, IsError, IsMissing, IsNull, IsNumeric, IsObject	v	v
Object Type	Is, TypeOf	v	

1 Not: Cdec and CVErr

See Also

Bits, Byte, Word, Int, and Large Operators and Keywords

Collection and Hash Keywords

Conversion Keywords

Dates and Times Keywords

Miscellaneous Keywords

Operators Keywords

String Manipulation Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Dates and Times Keywords

Action	Keywords	GB	VB6
Get the current date or time as a date	Date, Now, Time	v	v
Get the current date or time as a string	Date$, Now$, Time$ DateTime	v v	
Date to day, month, weekday, or year	Day, DayNo, Month, Weekday, Year	v	v
Time to hour, minute, or second	Hour, Minute, Second	v	v
Perform date calculations	DateAdd, DateDiff, DatePart	v	v
Return a date	DateSerial, DateValue	v	v
Return a time	TimeSerial, TimeValue	v	v
Extract Data and Timer	DateToDmy, DateToDmyHms TimeToHms	v	v
Set the date or time	Date, Time		v
Time a process	Timer	v	v
Performance	_TimerFreq,	v	

timer	Timer, oTimer, qTimer	
Performance processor timer	_RDTSC	v
C-time functions	_time, _ctime	v

See Also

Conversion Keywords

Data Types Keywords

Miscellaneous Keywords

Operators Keywords

String Manipulation Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Directories and Files Keywords

Action	Keywords	GB	VB6
Change directory and drive	ChDir, ChDrive	v	v
Return current path	CurDir	v	v
Return current directory and drive	_Dir, _Drive	v	
Copy/Move a file	FileCopy,	v	v
	CopyFile,	v	
	MoveFile	v	
Make/Remove directory or folder	MkDir, RmDir	v	v
Rename a file, directory, or folder	Name,	v	v
	Rename	v	
Return file date/time stamp	FileDateTime	v	v
Returns date/time	FileDateTimeAccess, FileDateTimeCreate	v	

Set date/time	SetFileDateTime, SetFileDateTimeAccess, SetFileDateTimeCreate, Touch	v	
Return and set file, directory, label attributes.	GetAttr, SetAttr	v	v
Returns and Set attribute information for a file	FGATTR, FSATTR	v	
File exists	Exist	v	
Return file length	FileLen,	v	v
	FileLen%	v	
Return file name or volume label	Dir	v	v
Dir stack	DirPush, DirPop, DirPopAll	v	
Long and short filename conversion	LongFileName, ShortFileName, LongPathName, ShortPathName, ShortProgName	v	
System directories and files	WinDir, SysDir, TempDir, TempFileName, KillTempFile	v	

See Also

Crypting, Mime encoding, Checksum Keywords

Data Types Keywords

Input and Output Keywords

Miscellaneous Keywords

String Manipulation Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Error Keywords

Action	Keywords	GB	VB6
Generate run-time errors	Clear, Error, Raise	v	v
Re-generate error	Throw	v	
Get error messages	Error$,	v	v
	Err$	v	
Get system error messages	SysErr	v	v
Provide error information	Err	v	v
Return Error variant	CVErr		v
Trap errors during run time	On Error, Resume	v	v
Trap errors during run time	Try/Catch/EndCatch	v	
Type verification	IsError	v	v
Verify integer value	Bound, BoundB, BoundC	v	

See Also

Compiler and Debug Keywords

Miscellaneous Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Graphical Keywords

In GFA-BASIC 32 graphical commands are performed directly on the form's client area. VB uses a second 'layer' to perform graphics methods like Line and Shape; they are therefore named 'controls'.

The Graphical keywords can be used on a **Form** and on the **Printer** objects.

Action	**Keywords**	**GB**	**VB6**
Color	Color, QBColor, ForeColor, BackColor	v	v
Color	BkColor, RGB, _RGB, GetRValue, GetGValue, GetBValue, PALETTEINDEX, PALETTERGB	v	
System colors	SysCol	v	
Clear screen	Cls	v	
Pen, brush	DefLine, DefFill	v	
Mode and background mode	DrawMode	v	v
	GraphMode	v	
Print	Print	v	v
	Locate, LocaYX, LocaXY, Print At, Vtab, Htab	v	

Text	DrawText, Text, TextXor, GrayText	v	
Line, rectangle	Line	v	v
(3d) Rectangles	Box, Pbox, Rbox, PRBox, Box3D, Pbox3D	v	
Circle, ellipse	Circle, Pcircle, Ellipse, Pellipse	v	
Set and get point	Pset, Plot, Draw, Line, SetDraw, Point, RGBPoint, PTst	v	
'Logo" drawing	SetDraw, Draw, QBDraw	v	
Bezier curve	Curve	v	
Polygon	PolyLine, PolyFill	v	
Scaling	ScaleMode, ScaleHeight, ScaleWidth, ScaleLeft, ScaleTop	v	v
Scaling extended	ScaleMMOO, ScaleMode$, ScaleMX, ScaleMY	v	
Conversion between scales	Scale(), ScaleX(), ScaleY()	v	
Drag rectangle	RubberBox, DragBox	v	
Rectangle intersection	rc_Intersect	v	
Clip	Clip	v	
Bitmaps	Get, Put, BitBlt, PatBlt, Stretch,	v	

	FreeBmp	
Fonts	Font, Font To, SetFont, GetFont, Rfont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont	v
GDI	GdiFlush	v
Himets, Pixel, Twips conversion	HimetsToPixelX, HimetsToPixelY, PixelsToHimetX, PixelsToHimetY, PixelsToTwipX, PixelsToTwipY, TwipsToPixelX, TwipsToPixelY	v

See Also

Miscellaneous Keywords

OCX/OLE Keywords

Window Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Input and Output Keywords

Action	Keywords	GB	VB6
Access or create a file	Open	v	v
Close files	Close, Reset	v	v
Flush data	Commit, Flush	v	
Control output appearance	Format, Print, Print, Spc, Tab	v	v
	Width #		v
Copy/Move a file	FileCopy	v	v
	CopyFile, MoveFile	v	
Get information about a file	EOF, FileAttr, FileDateTime, FileLen, FreeFile, GetAttr, Loc, LOF, Seek	v	v
Get information about a file	_File, TextEOF, RecordLOF	v	
Get and set	RelSeek, SeekEnd	v	
Get and set date/time	FileDateTimeAccess, FileDateTimeCreate, SetFileDateTime, SetFileDateTimeAccess, SetFileDateTimeCreate, Touch	v	
Manage files	Dir, Kill, Lock, Unlock, Name	v	v
Delete files and	KillFile, DeleteFile	v	

folders			
Manage files	Files	v	
Read from a file	Get, Input, Input#, Input?, LineInput#, Record	v	v
Return length of a file	FileLen	v	v
Set or get file attributes	FileAttr, GetAttr, SetAttr	v	v
Set or get file attributes	FSATTR, FGATTR	v	
Set read-write position in a file	Seek	v	v
	RelSeek, SeekEnd	v	
Write to a file	Print#, Put, , Record, Write	v	v
Block write/read to a file	Bput, Bget	v	
Block write/read a file	Bsave, Bload	v	
Read/write byte, word, or integer	Inp, Out	v	
Read/write string array	Recall, Store		
Read/write to PORT (byte, word, or integer)	Inp(Port), Out(Port)	v	
VB compatible, 32-bits functions	FileLen%, Loc%, LOF%, Seek%, RelSeek%	v	v

Note - All VB file functions operate with 32-bits integers, a value between 1 and 2,147,483,647 (equivalent to

2^31 - 1), inclusive.

All GFA-BASIC 32 file functions operate with 64-bit integers, by default. To use VB compatible commands use the functions with % postfix.

See Also

Arrays Keywords

Collection and Hash Keywords

Crypting, Mime encoding, Checksum Keywords

Data Types Keywords

Directories and Files Keywords

Miscellaneous Keywords

String Manipulation Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 01/03/2017 by James Gaite}

Math Keywords

Action	Keywords	GB	VB6
Integer arithmetic	Add, Sub, Mod, Mul, Div	v	
Floating point arithmetic	Fmod	v	
Derive trigonometric functions	Sin, Cos,Tan, Atn	v	v
Extended trigonometric functions	SinQ CosQ, Acos, Atan, Atan2, _hypot	v	
General calculations	Exp, Log, Sqr	v	v
GFA-BASIC 32 general calculations	Exp2, Exp10, Log2, Log10, Sqrt, Square, Pow, _j0, _j1, _jn, _y0, _y1, _yn	v	
Mantisse	LdExp, Mant, GetExp	v	
Generate random numbers	Randomize, Rnd	v	v
	Rand	v	
Generate random numbers C-style	_rand, _srand	v	
Get absolute value	Abs	v	v
Get the sign of an expression	Sgn	v	v
Perform numeric conversions	Fix, Int	v	v

Additional numeric conversions	Floor, Ceil, Trunc, Frac	v
Statistics	Variat, Combin, Permut	v
Rounding	Round, Fround, Qround	v
Minimum and maximum	Max, Min, MaxCur, MinCur, MaxI, MinI, iMax, iMin, MaxLarge, MinLarge	v
Odd and Even	Odd, Even	v
Incrementing and decrementing	Inc, Dec, Incr, Decr, Pred, Succ	v

See Also

Arrays Keywords

Bits, Byte, Word, Int, and Large Operators and Keywords

Collection and Hash Keywords

Conversion Keywords

Matrices Keywords

Miscellaneous Keywords

Operators Keywords

String Manipulation Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 17/10/2017 by James Gaite}

Matrices Keywords

Action	Keywords	GB	VB6
Arithmetic	Mat Add, Mat Sub, Mat Mul	v	
Copy, move	MatCpy, MatX Cpy, Mat Trans	v	
Initialize	Mat Clr, Mat Set, Mat One, Mat Neg	v	
Perform operations	Mat Det, Mat Qdet, Mat Rank, Mat Inv	v	
Read/write	Mat Print, Mat Read	v	
Normalize	Mat Norm	v	

See Also

Arrays Keywords

Bits, Byte, Word, Int, and Large Operators and Keywords

Data Types Keywords

Math Keywords

Operators Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Memory Keywords

Action	Keywords	GB	VB6
Memory management	cAlloc, mAlloc, mFree, mShrink, mReAlloc	v	
Memory initialization	MemBFill, MemWFill, MemLFill, MemSet, MemZero, ArrayFill	v	
Memory move/copy	Bmove, BlockMove, MemMove, MemCpy	v	
Memory And, Or, Xor	MemAnd, MemOr, MemXor	v	

See Also

Bits, Byte, Word, Int, and Large Operators and Keywords

Conversion Keywords

Data Types Keywords

Math Keywords

Miscellaneous Keywords

Operators Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Miscellaneous Keywords

Action	Keywords	GB	VB6
Process pending events	DoEvents	v	v
	Sleep, PeekEvent, GetEvent	v	
Pause	Pause, Delay	v	
Declare DLL functions	Declare	v	v
	FreeDll	v	
Run other programs	Shell	v	v
	System, Exec, WinExec, ShellExec	v	
Multimedia	Mci$, Mci, mciErr$, mciID	v	
Send keystrokes to an application	SendKeys	v	v
Sound	Beep,	v	v
	PlaySound	v	
Joystick	Stick, Strig	v	
System	Environ	v	v
	IsWinNT	v	
Provide a	Command,	v	v

command-line string	_dosCmd	v
Inline Assembler and Disassembler	Asm, DisAsm	v
Processor	GetRegs, _CPUID, _CPUID$, _CPUIDD, IsMMX	v
Thread	GetCurrentFiber, GetFiberData, GetTIB	v
Call a function through a pointer	C, LC, LP, Call, CallX, Ccall, LCCall, LpasCall, StdCall, LstdCall	v

See Also

Arrays Keywords

Bits, Byte, Word, Int, and Large Operators and Keywords

Collection and Hash Keywords

Control Flow Keywords

Compiler and Debug Keywords

Conversion Keywords

Crypting, Mime encoding, Checksum Keywords

Data Types Keywords

{Created by Sjouke Hamstra; Last updated: 17/10/2017 by James Gaite}

Operators Keywords

Action	Keywords	GB	VB6
Arithmetic	^, *, /, \, +, -	v	v1
Keywords	Add, Sub, Mul, Div, Fmod	v	
Keywords	Mod	v	v
Increment/decrement	++, --, Dec, Inc	v	
Assignment	+=, -=, /=, *=, %=, &=, |=, ^=	v	
Comparison	<, >, <>, ><, =<, <=, >=., =>, !=, =, ==, Is	v	v2
Floating point comparison	NEAR	v	
Logical operations	&&,, || ^^	v	
Logical NOT	!	v	
One's complement	~	v	
Bitwise operators	%&, |, %|, ^^	v	
32-bit Bitwise operators	And, Or, Xor, Imp, Eqv	v	v
64-bit Bitwise operators	And8, Or8, Xor8, Eqv8, Imp8	v	
Unary	*, V:,		

1 Not: ^

2 Not: ==; !=; ><; =<; =>

See Also

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

OCX/OLE Keywords

Action	Keywords	GB	VB6
Create OCX Form	Form, LoadForm, OpenW, ChildW, ParentW, Dialog	v	
Create OCX Controls	Ocx, OcxOcx, OcxScale	v	
Get OCX reference	Form(),OCX(), Me	v	
OCX Types	Form, Command, Option, CheckBox, RichEdit, ImageList, TreeView, ListView, Timer, Slider, Scroll, Image, Label, ProgressBar, TextBox, StatusBar, ListBox, ComboBox, Frame, CommDlg, MonthView, TabStrip, TrayIcon, Animation, UpDown	v	v
Type of an OLE object	TypeOf	v	
Set object reference	Set, New, Me, Nothing,	v	v
	Output	v	
OCX Mouse	MouseCursor, LoadCursor, MouseIcon	v	
OCX Collections	Buttons, CoumnHeaders, Panels, ListItems, Nodes, ListImages, Tabs, MenuItems, Forms, Controls	v	v
Disassembler	DisAsm	v	
Printer	Printer	v	v

Standard Objects	Font, StdFont, Picture, StdPicture	v	v
Picture objects	CreatePicture,	v	v
	LoadPicture, PaintPicture, SavePicture	v	
Informative Objects	App, Screen	v	v
Automation	CreateObject, GetObject	v	v
GUID	GUID, GUID$		

See Also

Collection and Hash Keywords

Conversion Keywords

Data Types Keywords

Errors Keywords

Miscellaneous Keywords

Variables and Constants Keywords

Window Keywords

{Created by Sjouke Hamstra; Last updated: 17/10/2017 by James Gaite}

Registry Keywords

Action	Keywords	GB	VB6
Delete program settings (VB)	DeleteSetting (VB) = vbDeleteSetting(GB)	v	v
Read program settings (VB)	GetSetting (VB) = vbGetSetting(GB)	v	v
Read all program settings (VB)	GetAllSettings		v
Save program settings (VB)	SaveSetting (VB) = vbSetSaving(GB)	v	v
Delete program settings (GB)	DeleteSetting	v	
Read program settings (GB)	GetSetting	v	
Save program settings (GB)	SaveSetting	v	
Open, create, and close keys	CreateRegKey, OpenRegKey, CloseRegKey	v	
Get value information	GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount	v	
Get sub key information	GetRegSubKey, GetRegSubKeyCount	v	

See Also

Bits, Byte, Word, Int, and Large Operators and Keywords

Conversion Keywords

Data Types Keywords

Memory Keywords

Miscellaneous Keywords

String Manipulation Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

String Manipulation Keywords

Action	Keywords	GB	VB6
Compare two strings	StrComp	v	v
	StrCmp, StrCmpI, LStrCmp, LStrCmpI	v	
Concatenation	$	v	
	+, &	v	v
Convert strings	StrConv		v
Convert to lowercase or uppercase	Format, Lcase, Ucase	v	v
	Upper, Lower	v	
Create string of repeating character.	Space, String	v	v
Find length of a string.	Len	v	v
Format a string	Format	v	v
	sprintf, Using	v	
Justify a string	Lset, Rset	v	v
Manipulate strings	InStr, Left, Ltrim, Mid, Right, Rtrim, Trim	v	v
Manipulate strings	RinStr, Mirror, SubStr, Ztrim	v	

Set string comparison rules.	Mode	v	v
Work with ASCII and ANSI values.	Asc, Chr	v	v
Translate	Xlate	v	
Replace	Replace		v
	reSub	v	
Regular expressions	Split, Join, preMatch, reMatch, ReSub	v	
Read a null-terminated string from memory	Char{}, CharPeek, Peek$, StrPeek	v	
Write a null-terminated string to memory	Char{}=, CharPoke, Poke$, StrPoke	v	

See Also

Arrays Keywords

Collection and Hash Keywords

Conversion Keywords

Crypting, Mime encoding, Checksum Keywords

Data Types Keywords

Dates and Times Keywords

Input and Output Keywords

Miscellaneous Keywords

Operators Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 17/10/2017 by James Gaite}

Variables and Constants Keywords

Action	Keywords	GB	VB6
Assign value.	Let, =	v	v
Clear variable	Clr, Clear, Erase	v	
Declare variables or constants.	Dim, Global, Local, Static, Const, Enum	v	v
Declare GUID constant	GUID	v	
Read data	Data, _Data, Read, Restore	v	
Get information about a variant.	IsArray, IsDate, IsEmpty, IsError, IsMissing, IsNull, IsNumeric, IsObject, TypeName, VarType	v	v
Get information about an OLE object	TypeOf	v	
Require explicit variable declarations.	Option Explicit	Always	v
Set default data type.	Deftype	v	v
Address of descriptor	ArrPtr, *	v	
Address of variable	VarPtr, V:, *	v	
Pointer Type	Pointer	v	
Procedure, Label address	ProcAddr, LabelAddr	v	
User-defined type	Type,	v	v

	Union	v
Size and offset of Type (elements)	SizeOf, BitSizeOf, BitOffsetOf, OffsetOf	v

See Also

Bits, Byte, Word, Int, and Large Operators and Keywords

Control Flow Keywords

Conversion Keywords

Data Types Keywords

Miscellaneous Keywords

Operators Keywords

String Manipulation Keywords

Variables and Constants Keywords

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

Window Keywords

Action	Keywords	GB	VB6
Windows creation	Form, OpenW, ChildW, ParentW, Dialog	v	
Get message	DoEvents	v	v
	Sleep, GetEvent, PeekEvent	v	
Control creation	Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, Ctext, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, Ltext, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, Rtext, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl	v	
Set window text	TitleW, _Win$	v	
Manage Windows	MoveW, SizeW, FullW, TopW, CloseW, ClearW, AdjustW, ShowW, DisableW, EnableW, Zoomed?, Visible?, Iconic?, ArrangeIcons	v	
Menu bar creation	Menu	v	
Menu bar	MenuItem	v	v

management			
Redirect output	Win, Output	v	
Get/Set window parameters	GetWinRect, WindGet, WindSet	v	
Menu() 16-bit support	Menu(), GetEvent, PeekEvent	v	
API messages	SendMessage, PostMessage, MakeWParam	v	
Information	GetDevCaps, SysMetric	v	
Mouse	MousePointer,	v	v
	DefMouse, HideM, ShowM	v	
Mouse capture	ReleaseCapture, SetCapture	v	
Mouse Input	Mouse, MouseX, MouseY, MouseK, MouseKB, MouseSX, MouseSY	v	
Keyboard Input	KeyGet, InKey, KeyTest	v	
Input dialog boxes	Prompt, InputBox	v	
Message boxes	Message, MsgBox, MsgBox0, Alert, AlertBox	v	
Context popup menu	Popup	v	
Atom API	_Atom$, Add, Atom Find, Atom Delete	v	

See Also

Graphical Keywords

Miscellaneous Keywords

OCX/OLE Keywords

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Functions, Procedures and Subs

Purpose

Blocks of code to which paramters or arguments can be passed and which perform one or more specific tasks; in the case of Functions, the result of any calculations can then be returned to the code that called it.

Evaluates an arithmetic expression which is repeatedly used throughout the program, whereby the result of the expression changes depending on the variables passed to it.

Syntax

Function[Var] name [**(**arglist**)**] [**As** type] [**Naked**]
[statements]
[**Exit Func**[**tion**] [**If**]]
[statements]
[name **=** expression | **Return** expression]
EndFunc[**tion**]

Proc[**edure**] name [(arglist)] [**Naked**]
[statements]
[**Exit Proc**[**edure**] [**If**]]
[statements]
EndProc[**edure**]

Sub name [(arglist)] [**Naked**]
[statements]
[**Exit Sub** [**If**]]

[statements]
EndSub

Description

Born of Procedural or structured programming introduced with the C language, these three types of subroutines are designed to perform specific tasks independent of other subroutines as well as clones of themselves.

Although the structure of all subroutines is similar, each has a specific purpose.

- **Procedures:** A Procedure, or *general procedure*, is a structure which tells the application how to perform a specific task and these form the main building blocks of the computational part of the program. Once a *general procedure* is defined, it must be specifically invoked by the application.
- **Subs:** A Sub, though similar to a Procedure, is designed to be an *event procedure*. This type of code usually has fixed parameters or arguments and, once defined, lies dormant until called upon to respond to events caused by the user or triggered by the system.
  Subs can also be used for porting VB sub routines and for this reason the default variable type of parameters is Variant; it is also possible to use Subs in a similar way to Procedures, although care should be taken due to the different default method of passing parameters (see below).
- **Functions:** Functions are considered by some to be the most important components of structured programming. They enable repeated evaluation of both arithmetic and string expressions and, unlike Procedures and Subs, Functions return the result of this evaluation, can be incorporated into expressions and

are the only type of subroutine that can sit on the right hand side of an operator (e.g. =, >=, !=, etc.). GFA-BASIC 32 supports two kind of functions: the GFA-BASIC 16, C/C++ compatible **Function** which takes parameters by value with default type Double, and **FunctionVar**, which is VB compatible, and, by default, passes parameters of type Variant *implicitly* **ByRef** (see below); both return a Variant unless otherwise specified.

The body of a subroutine is composed of the declaration (**Function**, **FunctionVar**, **Procedure** or **Sub**) with a name and parameter list, definition of local variables (**Local**...), the subroutine statements (with a return value for Functions) and a corresponding end marker (**EndFunction**, **EndProcedure** or **EndSub**).

name	Required. Name of the subroutine. This follows standard variable naming conventions.
arglist	Optional. List of variables representing arguments that are passed to the subroutine when it is called. Unless otherwise specified, parameters passed to **Function** and **Procedure** are Double and those passed to **FunctionVar** and **Sub** are Variant. Multiple variables are separated by commas and are entered in the following format: **ByRef** *varname*[()] [**As** *Type*] • Indicates that an argument is passed by reference. This means that the pointer to the argument is passed rather than the value, so that any changes made to the

argument inside the subroutine also affect the parent variable or array.
- All variable types and arrays can be passed ByRef, although to should be noted that Hash Tables are passed as a constant rather than a variable and can not be altered within the subroutine.
- Variables passed as arguments MUST match the argument variable type, otherwise an error is raised.

ByVal *varname* [**As** *Type*]

- Indicates that an argument is passed by value. This means that only the value of the argument is passed rather than the pointer, so that any changes made to the argument inside the subroutine do not affect the parent variable (see below regarding arrays).
- All variable types can be passed ByVal, but not Arrays, User Defined Types and Hash Tables; if you try and pass an array ByVal, GFABASIC-32 will send it ByRef instead; if you try and pass UDTs or Hash Tables, an error will be raised.
- Variables passed ByVal do not have to match the argument type as GFABASIC-32 will attempt to convert them; however, if a conversion is not possible (e.g. a String passed to a parameter of type Double), an error will be raised.
- In GFABASIC-32, this is the default state for **Function** and **Procedure** and thus the ByVal can be omitted.

Optional *varname* [**As** *Type*][= *defaultvalue*]

- Indicates that an argument is not required and can be omitted without raising an error. If no value is passed, then the argument assumes the value of *defaultvalue* if specified or, if not, zero for all numeric variables, Missing for Strings (see Known Issues) and Variants and Nothing for an Object; the only valid *defaultvalue* for an Object is Nothing.
- All parameters passed using the **Optional** keyword are considered to be passed **ByVal**, even in subroutines where parameters are **ByRef** by default.
- All 'non-complex' variable types (numeric, string, Variant and Object) can be passed, but not User Defined Types, Hash Tables nor Arrays of any type as these can only be passed ByRef in GFABASIC-32.

ParamArray *varname()*

- An extenstion of the Optional keyword, ParamArray allows the entry of an arbitrary number of optional parameters of any type that can be stored in as a Variant.
- ParamArray is an extremely flexible device and effectively allows you to customise the list of arguments to suit

different situations the subroutine may be called upon to handle.

- Due to the fact that there is no way of limiting how many parameters ParamArray is to pass, it should always be the last argument listed.
- All parameters passed using this method are considered to be passed **ByVal**, even in subroutines where parameters are **ByRef** by default.
- The array passed is technically an array of Variant types, the length of which is determined by the number of parameters entered, and functions such as UBound, LBound and Dim? can be used to determine its size; however, it is an OLE, not a GFABASIC-32, Array and can not be passed to another subroutine in its native form, only as an array in a variant as shown below:

```
test(1, 2, 3, 4)

Proc test(ParamArray p())
  Print "Dim?(p()) ="; Dim?(p())
  Local a As Variant : a = p :
    Print "a(1) ="; a(1)
  Print "Dim?(a) =";  UBound(a) + 1
    // Dim? does not always work
    with Arrays in Variants
  test2(a)
EndProc

Proc test2(ParamArray p())
  Print : Print "Dim?(p()) ="; Dim?
    (p())
  Print "p(0)(1) = "; p(0)(1)
EndProc
```

Also worth noting is that the array ignores the value of Option Base and has a lower boundary of 0 (zero).

- **Note:** There are two 'Known Issues' with **ParamArray**:
  1. Trying to pass a value from a numeric array fails and the value is passed as an Empty value instead; furthermore, all subsequent parameters are also passed as Empty. To get around this bug, pass any array values wrapped in an explicit OLE conversion function like **CLong**(), **CDbl**(), etc.
  2. There is an occasional compiler error (it happens sometimes but not always) if handles or non-integers are passed into the **ParamArray** array; if you get strange compiler errors, try changing all parameters to integers or wrapping them in an explicit OLE conversion function (**CLong**(), **CDbl**(), etc.) and this could well solve the problem.

statements	Optional. Any group of statements to be executed within the subroutine.
Naked	See here.

[**Function** and **FunctionVar** only] There are the two methods of returning a value from a Function:

- **name = expression** - This method assigns a value to the function name and any number of assignments can be made anywhere within the subroutine; the actual value returned is the last one to be assigned before the end of the Function is reached.
  The variable type of the returned value is determined by the type assigned to the function itself (in the **As** *Type* expression following the declaration of the Function) or is Double for **Function** or Variant for **FunctionVar**. The return type can also be determined by adding a postfix to the end of the Function name (e.g. Func$ will return a string, Func? a boolean) but keep in mind that Functions postfix-ed with a $ do not work in LG32 Libraries.
- **Return expression** - This method uses the **Return** command along with an expression of the value to return. Although many **Return** statements can be included within the code of a Function, the program will return a value on the first instance encountered and terminate the Function.
  The variable type of the return value is determined by the contents of the expression UNLESS the Function has been declared with a return type (see above), in which case the return variable type must be at least compatible with that type.

If no value is returned, the procedure returns a default value: a numeric function returns 0, a string function returns a zero-length string (""), a **Variant** function returns Empty and an Object returns Nothing.

[**FunctionVar** and **Sub** only] The default method for passing parameters for both these routines is *implicitly* by reference, which differs both from GFABASIC-32's default of **ByVal** and the *explicit* method of passing parameters by reference using **ByRef**. Basically, passing values *implicitly* by reference is a hybrid form used in VB: if the value being passed is a variable AND the type of that variable matches the declared type of the parameter, then the variable will be passed ByRef; otherwise, it will be passed ByVal. (Actually a copy is passed by reference, but the effect is similar to if it was passed ByVal. For a more in-depth description of how this all works, see here.)

It is important to keep this fact in mind when using these commands as, otherwise, you may find variables you intended to pass by value taking on changes made inside the called routine as they are actually being passed by reference; and, similarly, if you are passing a variable by reference but GFABASIC-32 does not recognise it as being exactly the same as that of the parameter, changes made within the subroutine will not be passed back to calling routine. Default behaviour can be changed - as with GFABASIC routines - by using **ByRef** and **ByVal**, but it is Strongly advised to use ByVal and/or ByRef explicitly or otherwise to stick to Procedure/Function subroutines.

Note that *implicit* referencing does not work with Arrays, User-Defined Types and Hash Tables: these are all sent by reference, the last two needing this to be *explicitly* stated by using **ByRef**, as with normal GFABASIC routines.

To better illustrate how *implicit* by referencing works, see the following example:

```
' FunctionVar 'default' passing matching variable
Local a = "George"
Print hellof(a)          // Prints "Hello George"
Print a : Print          // Prints "Hello George"
' FunctionVar 'default' passing non-matching
  variable
```

```
Local a$ = "George"
Print hellof(a$)          // Prints "Hello George"
Print a$ : Print          // Prints "George"
' FunctionVar 'default' passing literal
Print hellof("George")    // Prints "Hello George"
Print
' Sub 'default' passing matching variable
a = "George"
hellos(a)                 // Prints "Hello George"
Print a : Print           // Prints "Hello George"
' Sub 'default' passing non-matching variable
a$ = "George"
hellos(a$)                // Prints "Hello George"
Print a$ : Print          // Prints "George"
' Sub 'default' passing literal
hellos("George")          // Prints "Hello George"

FunctionVar hellof(a)
  a = "Hello " & a
  hellof = a
EndFunc

Sub hellos(a)
  a = "Hello " & a
  Print a
EndSub
```

To get a better idea of how this works, change the type of the parameter in both subroutines to '**As String**'.

The **Exit...** statements cause an immediate exit from a subroutine and program execution continues with the statement following that which called the subroutine. Any number of **Exit...** statements can appear anywhere in a subroutine and can be qualified with an optional **If** expression which determines whether the exit occurs or not.

Example

The following example asks you to select different shapes to draw and counts how many of each type you select; it will, however, not let you redraw the same shape you have just drawn.

```
Global ct%(1 To 4), lastshape%
Local wh% = WinHeight(370), ww% = WinWidth(230)
OpenW Fixed 1, 10, 10, ww%, wh% : Win_1.AutoRedraw
  = 1
Ocx Command cmd(1) = "Draw Circle", 10, 10, 100,
  25
Ocx Command cmd(2) = "Draw Square", 120, 10, 100,
  25
Ocx Command cmd(3) = "Draw Filled Circle", 10, 45,
  100, 25
Ocx Command cmd(4) = "Draw Filled Square", 120,
  45, 100, 25
DisplayCount(ct%())
Do : Sleep : Until Win_1 Is Nothing

Procedure DisplayCount(ByRef ct%(), Optional
  shape%)
  // ct%()  - A pointer to the 32-bit Integer array
    holding the current count stats - this is
    incremented 'in-procedure', which updates the
    parent array as well
  // shape% - The array element to increment
    dependent upon which shape was drawn; if not
    passed, then it defaults to zero.
  If shape% <> 0 Then Inc ct%(shape%)
  Text 10, 300, "Circles:" : Text 85, 300, ct%(1)
  Text 10, 315, "Squares:" : Text 85, 315, ct%(2)
  Text 10, 330, "Filled Circles:" : Text 85, 330,
    ct%(3)
  Text 10, 345, "Filled Squares:" : Text 85, 345,
    ct%(4)
EndProcedure

Procedure DrawShape(shape%, Optional filled? =
  True, ParamArray coords())
  // shape%    - A ByVal Int32 parameter describing
    the shape to be drawn
  // filled?   - An optional ByVal Boolean parameter
    (defaults to TRUE (-1) if not passed)
    determining whether the shape is filled or not
  // coords() - An array of additional optional
    parameters holding values for x%, y% and
    possibly x1%, y1% and r% depending upon shape%
  Local x%, y%, x1%, y1%, r%
  x% = coords(0), y% = coords(1)                 //
    Assign x% and y% from the first two values in
    coords()
  Select shape%
  Case 1 // Circle
    r% = coords(2)                               //
      Assign r% from coord(2)
    If filled? : PCircle x%, y%, r%
    Else : Circle x%, y%, r%
    EndIf
  Case 2 // Square
    If Dim?(coords()) = 4                        //
      If four parameters passed in coords()...
      x1% = coords(2), y1% = coords(3)          //
        ...then assign x1% and y1% from the third
        and fourth values...
    Else                                         //
      ...else...
      x1% = x% + 190 : y1% = y% + 190           //
        ...assume a width and height of 190 pixels.
    EndIf
    If filled? : PBox x%, y%, x1%, y1%
    Else : Box x%, y%, x1%, y1%
    EndIf
  EndSelect
EndProcedure

Sub      cmd_Click(Index%)
  // An 'event procedure' which is activated when
    one of the four command buttons is clicked
  // Index% - An expected or mandatory parameter
    which indicates which of the four command
    buttons was clicked.
  Exit Sub If Index% = lastshape%               //
    If selected shape the same as the last one, do
    not redraw and count
  Color $FFFFFF : DrawShape(2, , 20, 90, 210, 280)
    : Color 0
  Select Index%
  Case 1 : DrawShape(1, False, 115, 185, 95)
  Case 2 : DrawShape(2, False, 20, 90)
  Case 3 : DrawShape(1, , 115, 185, 95)
  Case 4 : DrawShape(2, , 20, 90)
  EndSelect
  lastshape% = Index
  DisplayCount(ct%(), Index%)
EndSub

Function  WinHeight(height%) As Int32
  // This function assigns the 32-bit integer value
    to the function name and returns this value
    when the function is complete
  WinHeight = height% + Screen.cyFixedFrame +
    Screen.cyCaption
EndFunction

Function  WinWidth(width%)
```

```
  // This function uses the Return command to
    return a 32-bit integer
  Return width% + (Screen.cxFixedFrame * 2)
EndFunction
```

Remarks

Subrountines can be recursive; that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow. For similar reasons, the **Static** keyword usually isn't used with recursive subroutines.

Always use **FunctionVar** for VB functions. Once the function works correctly, it is advisable to change it in a normal **Function** and change the default types as **FunctionVar** is time consuming due to the use of Variants.

Known Issues

- Errors can occur when the function name is used as the return value, as shown in the example below:

```
' Courtesy of 'Code Lab'
Type vector
  x As Double
  y As Double
EndType
Dim a As vector, b As vector, c As vector, d As
  vector
Print "a:", a.x, a.y
c = tovector(a, 1) : Print "c:", c.x, c.y
Print "b:", b.x, b.y
d = tovector(b, 0) : Print "d:", d.x, d.y
Print "d.x shouldn't have this value"

Function tovector(ByRef v As vector, which) As
  vector
  If which = 1
    tovector.x = 10
  Else
    tovector.y = 5
  EndIf
EndFunction
```

To work around this problem, use a locally defined variable as the return value instead as in the example below:

```
' Courtesy of Thomas Müller-Wirts
Type vector
  x As Double
  y As Double
EndType
Dim a As vector, b As vector, c As vector, d As
  vector
Print "a:", a.x, a.y
c = tovector(a, 1) : Print "c:", c.x, c.y
Print "b:", b.x, b.y
d = tovector(b, 0) : Print "d:", d.x, d.y
Print "d.x is now zero as it should be"

Function tovector(ByRef v As vector, which) As
  vector
  Local vr As vector
  If which = 1
    vr.x = 10
  Else
    vr.y = 5
  EndIf
  Return vr
EndFunction
```

- There is an obscure error involving a Boolean passed to Variant parameters from procedures containing a **Gosub...Return** construct. See here for more details.

- With most Optional Parameters, if a value is not sent then a temporary variable is created within the receiving subroutine which can then be referenced and changed as required; this, however, is not the case when the optional parameter is a **String**: in that instance, only a Null pointer is sent so any attempts to reference or change the variable - barring checking its status with **IsMissing** - returns an Access Error as shown by the example below:

```
Print Test(3)

Function Test(a, Optional s$)
  If IsMissing(s$)
    s$ = "No value" // Access Error on this line
  EndIf
  Return s$
EndFunction
```

The best way to get round this oddity is either by judicious use of the **IsMissing** function as below...

```
Print Test(3)
Print Test(3, "A Value")

Function Test(a, Optional s$)
  If IsMissing(s$) Return "No Value"
  Return s$
EndFunction
```

or creating a temporary string within the subroutine and using it as follows:

```
Print Test(3)
Print Test(3, "A Value")

Function Test(a, Optional s$)
  Local sv$ = Iif(IsMissing(s$), "No Value", s$)
  Return sv$
EndFunction
```

This last example also demonstrates how to get round the inability to set a default value to an Optional String parameter.
[Reported by James Gaite, 05/08/2019]

{Created by Sjouke Hamstra; Last updated: 05/08/2019 by James Gaite}

GoSub Command

Purpose

Branches to and returns from a subroutine within procedure.

Syntax

GoSub label
...
label:
...
Return

Description

You can use **GoSub** and **Return** anywhere in a procedure, but **GoSub** and the corresponding **Return** statement must be in the same procedure. A subroutine can contain more than one **Return** statement, but the first **Return** statement encountered causes the flow of execution to branch back to the statement immediately following the most recently executed **GoSub** statement.

You can't enter or exit **Sub** procedures with **GoSub...Return**.

Example

```
OpenW 1
test_mark // call Procedure
Do : Sleep : Until Me Is Nothing
CloseW # 1
```

```
Procedure test_mark
  Text 50, 20, "Hallo"
  GoSub mar1
  GoSub mar2
  GoSub mar3
  Text 250, 50, "GmbH"
Return
  mar1:
  Text 50, 50, "GFA"
Return
  mar2:
  Text 80, 50, "Software"
Return
  mar3:
  Text 160, 50, "Technologies"
EndProc
```

Known Error

There is an obscure error involving a Boolean passed to Variant parameters from procedures containing a **Gosub...Return** construct. See here for more details.

Remarks

A label might consist of a number (10) or start with alphanumeric character followed by more characters and ended with a semi-colon (p2:).

The label has function scope and cannot be redeclared within the function. However, the same name can be used as a label in different functions.

See Also

On Gosub, Goto

{Created by Sjouke Hamstra; Last updated: 11/03/2018 by James Gaite}

ReDim Command

Purpose

Reallocates storage space for a dynamic array.

Syntax

ReDim *varname***(***subscripts***)** [**As** *type*] [, *varname***(***subscripts***)** [**As** *type*]] **. . .**

varname	*: variable name*
subscripts	*: dimensions of an array*

Description

The **ReDim** statement is used to size or resize a dynamic array that has already been formally declared using a **Global**, **Public**, **Local** or **Dim** statement with or without empty parentheses (without dimension subscripts).

You can use the **ReDim** statement repeatedly to change the number of elements and dimensions in an array. However, you can not declare an array of one data type and later use **ReDim** to change the array to another data type.

ReDim does not clear the elements from the array, so data will remain in the elements that exist before and after redim-ing. To explicitly erase all elements use **Erase** before redim-ing, then no old data will be passed to the new redim-ed array.

Example

```
Dim MyArray() As Integer    ' Declare dynamic
  array.
ReDim MyArray(5)           ' Allocate 5 elements.
Local a() As String
ReDim a(10 To 50)
Erase a() : ReDim a(10 .. 50)
ReDim a(10 ... 50, -1 To 9)
```

Remarks

Dim a() As Int : ReDim a(count) requires an additional 72 bytes of (stack) memory compared to Dim a(count).

GFA-BASIC 32 doesn't provide the Preserve keyword, because the data of the array isn't erased.

An array in a Variant cannot be redimmed:

```
Dim v = Array(1, 2, "Hello") : ReDim v(5) // not
  possible
```

Known Issues

An array, when first dimensioned, and unless otherwise stated, takes its lower boundary (or **LBound**) from the value of **Option Base**; however, when that array is **ReDim**'ed, both the existing **LBound** value and the value of **Option Base** are ignored and the lower boundary is set to zero, unless explicitly set otherwise within the **ReDim** statement. This is shown in the example below:

```
Option Base 1
Dim a%(10), b$(1 To 10), c&(-4 To 100)
Print LBound(a%()), LBound(b$()), LBound(c&()) //
  Prints 1  1 -4
ReDim a%(10), b$(10), c&(100)
```

```
Print LBound(a%()), LBound(b$()), LBound(c&()) //
  Prints 0  0  0
```

This can cause some odd effects, such as when sorting the array in question. To get around this bug, if an array was dimensioned with a lower boundary other than zero (through **Option Base** or otherwise), then, when redim'ing, explicitly specify the lower boundary in the **ReDim** statement like this:

```
ReDim a%(1 To 10), b$(1 To 10), c&(-4 To 100)
Print LBound(a%()), LBound(b$()), LBound(c&()) //
  Prints 1  1 -4
```

See Also

Dim, Dim?, Erase

{Created by Sjouke Hamstra; Last updated: 04/03/2017 by James Gaite}

Iif Function and ?: Operator

Returns one of two parts, depending on the evaluation of a condition.

Syntax

result = **Iif**(condition, truepart, falsepart)

result = (condition **?** truepart **:** falsepart)

condition	*: boolean expression*
truepart	*: Value or expression returned if condition is True.*
falsepart	*: Value or expression returned if condition is False.*

Description

The **Iif** function and **?:** operator are synonymous and can be used as a shortcut for an I**f...Else** statement. It is typically used as part of a larger expression where an **If...Else** statement would be awkward; however, both constructs have limitations (see Known Issues below). .

Example

```
Local greeting$ = "Good" + Iif( Hour(Now) > 17, "
 evening.", " day.")
Print greeting$
```

Or using the **?:** operator:

```
Local greeting$ = "Good" + ( Hour(Now) > 17 ? "
  evening." : " day.")
Print greeting$
```

Both examples create a string containing "Good evening." if it is after 6pm. The equivalent code using an **If...Else** statement would look as follows:

```
Local greeting$ = "Good"
If Hour(Now) > 17
  greeting += " evening."
Else
  greeting += " day."
EndIf
Print greeting$
```

Known Issues

Using user-defined functions in the *truepart* and *falsepart* elements of this expression can sometime return an EdCodeGen error; use the If...Else construct if this occurs.
[Reported by James Gaite, 03/02/2015]

Alternatively, you could create a masking function to get around the problem like so:

```
Function  IifX(cond?, v1 As Variant, v2 As
  Variant)
  Return Iif(cond?, v1, v2)
EndFunction
```

See Also

If...Else

{Created by Sjouke Hamstra; Last updated: 18/10/2017 by James Gaite}

Choose Function

Purpose

Selects and returns a value from a list of arguments.

Syntax

Choose(index, choice-1[, choice-2, ... [, choice-n]])

Description

Choose returns a value from the list of choices based on the value of index. If index is 1, **Choose** returns the first choice in the list; if index is 2, it returns the second choice, and so on.

You can use **Choose** to look up a value in a list of possibilities. For example, if index evaluates to 3 and choice-1 = "one", choice-2 = "two", and choice-3 = "three", Choose returns "three". This capability is particularly useful if index represents the value in an option group.

Choose evaluates only the choice of the given index!

The **Choose** function returns a Null if index is less than 1 or greater than the number of choices listed.

If index is not a whole number, it is rounded to the nearest whole number before being evaluated.

Example

```
Local Byte ch, n
```

```
For n = 1 To 3
  Print "When ch = "; n; ", "; Choose(n, "First",
    "Second", "Third"); " will be printed."
Next n
```

Remarks

The VB function **Choose** evaluates every choice, GFA-BASIC 32 only the specified one.

See Also

Iif

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Date Function

Purpose

Returns the system date as a **Date** type.

Syntax

d = **Date**

d: Date expression

Description

Example

Dim MyDate As Date

MyDate = Date ' MyDate contains the current system date.

Remarks

A comparison operation with **Date** is possible only when both sides are of data type **Date.** When necessary cast the left- or right side using a **CDate**.

```
Local datum$ = Format(Date, "dd/mm/yyyy")
Local datum% = CDate(datum$)
Debug.Show
Trace datum$
Trace datum%
Trace Date
Trace Date$(datum%)
Trace Date = datum%
Trace Date = CDate(datum%)
Trace Date = CDate(datum$)
```

See Also

Now, Time

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

Pointer Keyword

Purpose

Pointer is a data type to declare variables as pointers.

Syntax

Dim p **As** [**Register**] **Pointer** [**To**] type

Pointer p = addr%

addr% = **Pointer**(p)

p:pointer variable
type:any data type
addr%:memory address

Description

The **Pointer** command and the **Pointer**() function apply to a variable of the **Pointer** [**To**] data type. A pointer variable can be declared for any type. The pointer variable then behaves as a variable of that type, but not before the pointer variable is assigned an address. Remember that each variable denotes a memory address of a specific size. An **Integer** variable holds the address of a 4 byte of memory block to store a value. In the same way, a pointer variable must be assigned a piece of memory to store the data type's value. The assignment of an address is done with the command **Pointer** p = addr%. The reverse, to get the address of memory pointer p points to, is done using the function addr% = **Pointer**(p).

In the next example a pointer to a **Double** data type is declared and assigned a memory location of 8 bytes in size. After the pointer assignment, the variable behaves like a **Double**. To check it, the variable is assigned the value 3.14, and then the memory location is peeked.

```
Dim addr% = mAlloc(8)
Local pdbl As Pointer Double
Pointer pdbl = addr%
pdbl = 3.14
Print DblPeek(addr%) // Prints 3.14
~mFree(addr%)
```

Pointers are more interesting used with user-defined types. In particularly, pointers are inevitable with API functions and messages that hand over pointers to structures (Type). For example, the WM_NOTIFY message used with notification messages from common controls specifies a pointer to the NMHDR type in the lParam parameter of the message. To get access to the type elements the address must be assigned to a variable of **Pointer To** NMHDR. The Example 1 shows how this is done.

Another use for pointers is for linked lists. A double linked list might use the following user-defined type:

```
Type LLIST
  pNext As Pointer To LLIST
  pPrev As Pointer To LLIST
  value As Int
EndType
Global MyList As LLIST
Global pList As Pointer To LLIST
Pointer(pList) = V:MyList
```

The pNext and pPrev elements should be assigned memory addresses using Pointer pNext =.

```
Pointer(pList.pNext) = mAlloc(SizeOf(LLIST))
Pointer(pList.pPrev) = Pointer(pList)
Pointer(pList) = Pointer(pList.pNext)
pList.value = 2
```

Pointer arithmetic differs from C/C++, where the size of the type of the pointer is automatically included. In GFA-BASIC 32 incrementing a pointer involves adding the size of the type explicitly.

```
Pointer(p) = Pointer(p) + SizeOf(p) * 1
```

Example

Example 1

```
Sub frm_MessageProc(hWnd%, Mess%, wParam%,
  lParam%, retval%, ValidRet?)
  Dim hdr As Pointer NMHDR
  Switch Mess
  Case WM_NOTIFY
    Pointer(hdr) = lParam
    Print hdr.idfrom
  EndSwitch
EndSub
Type NMHDR
  hwndFrom As Long
  idfrom As Long
  code As Long
EndType
```

Example 2

```
Debug.Show
Local a$ = "12345", x%
Local aa As Pointer To Int
```

```
' Assign a memory location to the Integer:
Pointer aa = V:a$
Trace Pointer(aa)
Trace V:a$
Trace V:aa
'************
Trace aa
'************
Trace a$
' Change the contents of a$
aa = $41424344
Trace a$
' Add one to the pointer
Pointer aa = Pointer(aa) + 1
' change the contents from the second position
aa = $41424344
Trace a$
```

Remarks

In GFA-BASIC 32 a user-defined variable may have the same name as the Type name. In VB or C/C++ this not allowed.

Note GFA-BASIC 32 provides s a double linked list with the **Hash** data type.

See Also

Hash

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Hash Table

Purpose

Provides a hash table, which maps string keys to values.

Syntax

Dim ht **As Hash** Type

ht	*: varname*
Type	*: simple type*

Description

A **Hash** is a one dimensioned 'array' and its index is a string. A Hash table has no size, just a number of elements. For Example, with **Dim** ht **As Hash Int** declares a dynamic table (array) *ht* of type **Int**, which has initially no elements.

Type specifies the data type of the values to be stored in the hash table. *Type* may be **Byte**, **Boolean**, **Card**, **Short**, **Word**, **Integer**, **Long**, **Large**, **Currency**, **Single**, **Double**, **Date**, **String** (variable-length strings only) and **Variant.**

A hash table supports the creation, storage, and retrieval of key/value pairs in memory. The hash table maps string keys to values; the index for hash tables is a string. When the type of the hash table is String, the hash table acts as a dictionary (string/string pair).

After declaring a hash, you can add elements of appropriate type to it, in one of two ways:

```
Hash Add ht["key"], value
ht["key"] = value
```

A **Hash** uses brackets to access an element. To access an element stored in a hash table, you simply specify the *key*.

```
value = ht["key"]
```

To obtain the number of elements of the hash table you use **%** as the key:

```
number_of_elements = ht[ % ]
```

To inquire if an element exists in the table prefix the key with a question mark:

```
If Not ht[? "key"] Then Hash Add ht["key"], value
```

A key cannot be duplicated, all keys must be unique. When you add a value with an already existing key, the value isn't added and an error is generated.

Internally, the Hash is implemented as a double linked list. Each element has its own position (index) in the list. Therefore, a hash table can be accessed as if it were an array using a number as the index starting at 1. To mark a key as numeric value, rather than a string, precede the numeric value with **%**.

```
value = ht[% 1]
```

In the same way you can iterate over a hash table using **For Next**:

```
For i = 1 To ht[ % ]
 Print ht[ % i]
Next
```

Another way to iterate over a hash table is by using **For Each** *element* **In** *hashvar*.

```
Dim e As Int  // same type as Hash <type>
For Each e In ht
  Print "Element = "; e; " at position "; Each; "
    and Key = "; ht[$ Each]
Next
```

The variable *e* receives the value from that position and must be of the same type as the type of the Hash. **Each** returns the current position (index) in the hash table.

A hash table can also store values without a key. The Hash is reduced to a double linked list, what it basically is. The only way to add key-less elements to the list is by using **Hash Add.** An advantage of **Hash Add** is that it gives control on the position of the elements.

Hash Add ht[], *value*

Hash Add ht[] **Before** idx, *value*

Hash Add ht[] **After** idx, *value*

To iterate over the list you can use **For Next** or **For Each**. The list variant of the hash table is used in quite some GFA-BASIC 32 commands like **Split**, **Join**, **Eval, reSub, reMatch**.

To remove an element use **Hash Remove** ht["key" | **%** index]

A hash table can be sorted by its keys, saved, loaded, and erased. The following commands are available:

Hash	Deletes the entire hash table.

Erase	
Hash Sort	Sort the hash table by its keys using the Mode Compare setting.
Hash Write	Save a hash table as an ASCII file.
Hash Input	Load a hash table from an ASCII file.
Hash Save	Save a hash table in binary format.
Hash Load	Load a hash table in binary format.

Other operations on a hash table are performed using the subscription notation [ ].

Subscript	**Meaning**
[%]	returns the number of elements
[% i **]**	accesses the element by index i (position)
[k$ **]**	accesses the element by key k$ (string)
[$ i **]**	returns the key for the given index i.
[? k$ **]**	returns a Boolean indicating whether the key k$ exists.
[?% i **]**	returns a Boolean indicating whether the index i exists.
[# k$ **]**	returns the index for the key k$.
[$$** k$ **]	returns the correct key string (upper and lower) for key k$.

Example

Example 1

```
Dim a As Hash String, e As String
Hash Add a[], "a"
Hash Add a[], "b"
Hash Add a[], "c"
Hash Add a[], "d"
a[% a[%]] = "e"         // replace last element
a[% 1] = "1"            // replace first element
a[ "Key" ] = "f"        // add element
Print a[$$ "key"]       // prints Key
For Each e In a[]       // iterate over table
  Print "(Pos)" & Each, " (element) " & e
Next
```

Example 2

```
Dim ha As Hash Int
// The value 27 will be assigned to element 1 with
  the key "a" and the index 1
ha["a"] = 27
// The value 29 will be assigned to element 2 with
  the key "xyz" and the index 2
ha["xyz"] = 29
// Output of element with index number 1:
Trace ha[% 1]           // Prints 27
// Output by using the key "a":
Trace ha["a"]           // Prints 27
// Output of element with index number 2:
Trace ha[% 2]           // Prints 29
// Output by using the key "xyz":
Trace ha["xyz"]         // Prints 29
// To get the key for the second element:
Trace ha[$ 2]           // Prints xyz
// The only way to add a value without using a key
Hash Add ha[], 100
// Call Proc dst to iterate through the Hash Table
dst(ha[])
// Check is key "a" exists:
Trace ha[? "a"]         // Prints True
// Check the correct formatting of key "a":
```

```
Trace ha[$$ "A"]          // Prints a
// Check to see if 3 is a valid index or not:
Trace ha[? % 3]           // Prints True
// To get the index which corresponds to key "xyz"
Trace ha[# "xyz"]         // Prints 2
// Request the number of elements in the Hash ha[]
Trace ha[%]               // Prints 3
Debug.Show

Procedure dst(ByRef h As Hash Int)
  // Remember h is read-only
  Local i%, j%
  For Each i% In h[]
    Debug.Print i%, Each, h[$ Each]
  Next
EndProc
```

Remarks

A **Hash** is passed **ByRef** to subroutines. Unfortunately, the argument is a const variable, so that the **Hash** is read-only and can't be modified.

Known Issues

Although the initial documentation says that it is possible to create a Hash Table full of **Object**s, this does not seem to be the case as there seems to be no way to assign the objects to the table. If the code below is run, a Syntax Error is thrown.

```
Dim a As Hash Object
Dim b As Object
Hash Add a["key"], b
```

[Reported by James Gaite, 15/02/2017]

In addition, although syntactically it is possible to add a Hash Table as a property of a User-defined Type, there is no means of accessing it and, as UDTs in GFABasic are static, there is no room for the Hash to expand to accept new values.
[Reported by Garibaldi, 16/11/2016]

See Also

Hash Add, Hash Erase, Hash Input, Hash Load, Hash Remove, Hash Save, Hash Sort, Hash Write

{Created by Sjouke Hamstra; Last updated: 04/03/2017 by James Gaite}

Split Command

Purpose

Splits a string into a **Hash String** (array of strings) by separating the string into substrings using regular expressions.

Syntax

Split hs[] = sexp, pattern [, max]

hs[]:Hash String
sexp, pattern:sexp
max:iexp

Description

Split returns the substrings of *sexp* in a **Hash String**, a list of strings. The **Hash** can be iterated over using **For Each** or a simple **For Next**. *pattern* is the string used to identify substring limits using a regular expression. The optional argument *max* specifies the maximum number of substrings to return. Although *pattern* can be a complex regular expression, it can also be a simple string that defines where the splits take place. For instance, the following example splits a string at a space character:

```
Dim hs As Hash String, s As String
Split hs[] = "This is a test", " "
For Each s In hs[]
  Debug Each ` s
Next
Debug.Show
```

This prints in the Output window:

1 This
2 is
3 a
4 test

Now, if the string contained a double space before the word test, there would be five elements found of which the fourth is an empty string.

```
Dim hs As Hash String
Split hs[] = "This is a  test", " "
Debug hs[%]' prints 5
Debug.Show
```

Often, this is not wanted. In that case regular expressions solve the problem elegantly. The pattern argument can be changed to a group of spaces: "[ ]+". See reMatch for an overview of the patterns.

```
Split hs[] = "This is a  test", "[ ]+"
```

Example

```
Local h As Hash String, s As String
Split h[] =
  "name,surname;street,12,"#9",Cologne,,Fax: 0111-
  1234567 ", "\s*[,;]\s*"
For Each s In h[]
  Debug Each`s
Next
Debug.Show
```

// this command splits the string into

```
h[% 1] = "name"
h[% 2] = "surname"
h[% 3] = "street"
h[% 4] = "12"
h[% 5] = ""
h[% 6] = "Cologne"
h[% 7] = ""
h[% 8] = "Fax: 0111-1234567"
```

The example shows a summary of personal information in a string separated by spaces, commas, tabs, and other white spaces. The search *pattern* is defined as: any number of spaces and/or tabs, then a comma or a semicolon, then again optional spaces or tabs. The string is separated with empty strings for missing details (commas, succeeding one another, with or without spaces or tabs between).

Remarks

The VB function ar$() = Split(*sexp*[**,** *delimiter*[**,** *max*[**,** *compare*]]]) is easily converted to GFA-BASIC 32. Rather than a string array, GFA-BASIC 32 uses a **Hash String**.

VB Split Function	***GFA-BASIC 32 Split Command***
Dim ar() As String Dim sexp As String Dim delim$ = " " ar = Split(sexp, delim, ,0) For i = 0 to UBound(ar()) Print ar(i) Next	Dim hs As Hash String Dim sexp As String Dim delim$ = " " Dim Cmp = Mode(Compare) Mode Compare 0 Split hs[] = sexp, delim Mode Compare Cmp For i = 1 To hs[%] Print hs[% i] Next

The *delimiter* argument may contain only one character in VB. The *compare* mode indicates the kind of comparison to use when evaluating substrings. In GFA-BASIC 32 use the **Mode Compare** before executing the **Split** command. After executing the **Mode Compare** should be restored.

See Also

Join, reMatch, reSub, preMatch, Hash

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Join Command

Purpose

Returns a string created by joining a number of substrings contained in a hash (array).

Syntax

Join strvar = hashvar[], delimiter

Description

With **Join** all elements of a **Hash String** are joined together, separated with *delimiter.*

Delimiter is the string (character) used to separate the substrings in the returned string. If delimiter is a zero-length string, all items in the list are concatenated with no delimiters.

Example

```
Dim h As Hash String
Dim s$ = "name,vor,str,12,,Köln,,Fax:0111-123467"
Split h[] = s$, ","
Clr s$
Join s$ = h[], ","
Print s$
```

This is the same as

```
Dim h As Hash String
Dim s$ = "name,vor,str,12,,Köln,,Fax:0111-123467"
```

```
Split h[] = s$, ","
Clr s$
Dim i%
s$ = ""
For i% = 1 To h[%]
  s$ = s$ + h[% i]
  If i != h[%] s$ = s$ + ","
Next
Print s$
```

See Also

Split, Hash

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

Eval Function

Purpose

Evaluates an expression at runtime.

Syntax

# = **Eval**(exp)

# = **Eval**(exp, hash[])

# = **Eval**(exp, [hash[]] , function)

exp: sexp
hash: Hash
function:Function

Description

Evaluates a formula or expression that is in the form of text and returns the result as a Double.

```
Print Eval("1+2*3")
Print Eval("1.045 ^ 20")
```

Eval knows the basic mathematical rules. The function **Eval** is performed by a small internal compiler. **Eval** accepts the following numbers, operators, and functions:

numbers	like with Val, also #date# or Hex with $x or binary with %x.
brackets	( )
addition	+

subtraction	-
multiplication	*
division	/
Modulo	Mod
Integer division	Div
leading sign	-
leading sign	+
raise to a higher power	^
raise to a higher power	**
Pi	3.1415..
goniometric functions	Sin(x), Cos(x), Tan(x)
logarithm	Log(x), Exp(x)
binary And	And
binary Or	Or
binary exclusive Or	Xor
Extreme values	Min(x, y), Max(x, y)
Diverse	Int(x), Trunc(x), Frac(x), Fix(x) Floor(x), Ceil(x), Abs(x), Sgn(x) Rnd, Rnd(x), Random(x), Rnd(x) is exact like Random(x), Sqr(x), Not, Fact(n), Combin(n, k), Permut(n, k), Exp2(x), Exp10(x), Log2(x) == Lb(x), Log10(x) == Lg(x) Log(x) == Ln(x)
Comparison*	< <= =< != <> >< = == >= => >
Not*	!

* The return value of a comparison, and of the ! - operator, are the floating point numbers -1.0 or 0.0.

Strings passed to the **Eval** function must be correctly formatted, otherwise they will throw an error (see Known Issue).

# = Eval(exp, hash[])

The second form accepts a **Hash Double** which holds variable values. The key of a Hash element is the variable name and the Double a value. The Hash will be evaluated before anything else, this excludes the usage of h["pi"], h["Sin"] or h["Mod"] keys.

# = Eval(exp, [hash[]], function)

The third form accepts, optionally, a **Hash Double** holding variable values and a function name. The *function* is executed for the string expression *exp* and the entire string expression *exp* will be passed as the first argument, followed by the number of parameters in *exp,* followed by the parameters itself in a one dimensional array of type double.

```
Function func(x$, n%, f#()) As Double
```

For example, suppose

```
Print Eval("new(1.3, 3*9)", , EvalFunc)
```

The **Eval** function executes *EvalFunc* passing "new" in x$, 2 in n%, and the arguments of new() in f(1) = 1.3, f(2) = 27. The *EvalFunc* could look like this:

```
Function EvalFunc(x$, n%, f#()) As Double
  Select Lower(x)
  Case "new"
    If n% = 2 Then Return Mul(f(1), f(1))
    Err.Raise 1001, "EvalFunc", "New(x, y) expects
      2 parameters, not " & n
  EndSelect
  Err.Raise 1000, "EvalFunc", " extension " & x$ &
    " unknown."
EndFunc
```

The maximum number of parameters for a user-defined function is 5. The parameter array is a 'global' array with a dimension of (1..10).

Example

The use of a Hash

```
OpenW 1
Local h As Hash Double, x%
h["a"] = 123
h["rent"] = 1.075
Print Eval("a / 7", h[])
Print Eval("10000*rent^30", h[])
```

Remarks

The performance of the **Eval** function depends on the parsing of the expression string and can hardly be compared by normal calculations. The following example shows the difference.

```
OpenW 1 : Win_1.FontName = "Courier New"
Local t#(3), d#, a#, b#, c#, i%
t(1) = Timer
For i = 1 To 100000
  d = Eval("1*2+3")
Next
t(1) = Timer - t(1)
t(2) = Timer
```

```
a = 1 : b = 2 : c = 3
For i = 1 To 100000
  d = a * b + c
Next
t(2) = Timer - t(2)
t(3) = Timer
For i = 1 To 100000
  d = Val("1") * Val("2") + Val("3")
Next
t(3) = Timer - t(3)
Print "Time for Eval:      "; t(1); Tab(45); " ~";
  Int((t(1) / t(2)) + 0.5); "times slower than
  variables"
Print "Time for Variables: "; t(2)
Print "Time for Val():     "; t(3); Tab(45); " ~";
  Int((t(3) / t(2)) + 0.5); "times slower than
  variables & ~"; Int((t(1) / t(3)) + 0.5); "times
  faster than Eval"
Do
  Sleep
Loop Until Me Is Nothing
```

The version with **Eval** is three times slower as the one with **Val**, and 200 times slower as the version with the variables.

Known Issue

If a string passed to **Eval** contains an invalid symbol (such as starting with an equals sign (=)), this will cause the program to stop BUT an error will not necessarily be thrown (sometimes an Invalid Parameter error appears, sometimes not).

See Also

Val

{Created by Sjouke Hamstra; Last updated: 17/10/2017 by James Gaite}

reMatch Function

Purpose

Searches a string expression or string array for occurrence of a substring using regular expressions.

Syntax

n = **reMatch**(sexp, pattern [, hash[] | address%])

n = **reMatch**(array$(), pattern ,from ,to [, hash[]])

sexp, pattern:string expression
address, from, to:iexp
hash[]:Hash String or Hash Int
array():string array
n:iexp

Description

In the simplest form **reMatch** searches a substring in a string like **InStr**(). The return value gives the position of the substring *pattern* within the string *sexp*. n = 0 when the substring isn't found. However, using regular expression patterns **reMatch** is capable of locating much more. For instance, an A followed by a b or d, then an e, and maybe a r. Next a point, comma, or a space, or an end-of-line.

"A[bd]er?([.,]|$)"

Special characters and sequences are used in writing patterns for regular expressions. The following table describes these characters and includes short examples showing how the characters are used.

Character	**Description**
\	Marks the next character as special. \\. a point; \\\\ a backslash; \* star; \\+ plus; \\[; \\]; \\(; \\); \\^; \\$. Any character that has a special meaning for a pattern.
^	Matches the beginning of input or line.
$	Matches the end of input or line.
*****	Matches the preceding character zero or more times. "zo*" matches either "z" or "zoo."
+	Matches the preceding character one or more times. "zo+" matches "zoo" but not "z."
?	Matches the preceding character zero or one time. "a?ve?" matches the "ve" in "never."
.	Matches any single character except a newline character.
(*pattern***)**	A group. To match parentheses characters ( ), use "\\(" or "\\)".
*x***|***y*	Matches either *x* or *y*. "z|food?" matches "zoo" or "food."
[*xyz***]**	A character set. Matches any one of the enclosed characters. "[abc]" matches the "a" in "plain." The special characters (,), *, ., $ and ^ have no special meaning inside a set.
[^*xyz***]**	A negative character set. Matches any character not enclosed. "[^abc]" matches the "p" in "plain."
\\A	Matches the beginning of input or line, same as ^.
\\Z	Matches the end of input or line, same as $

\e	Matches a an escape character (Esc)
\cX	Matches a control character \cA (control-A)
\d	Matches a digit character. Equivalent to [0-9].
\D	Matches a nondigit character. Equivalent to [^0-9].
\f	Matches a form-feed character.
\n	Matches a linefeed character.
\r	Matches a carriage return character.
\s	Matches any white space including space, tab, form-feed, and so on. Equivalent to [\f\n\r\t\v]
\S	Matches any nonwhite space character. Equivalent to [^ \f\n\r\t\v]
\t	Matches a tab character.
\v	Matches a vertical tab character.
\w	Matches any word character including underscore. Equivalent to [A-Za-z0-9_].
\W	Matches any nonword character. Equivalent to [^A-Za-z0-9_].
\*num*	Matches *num*, where *num* is a positive integer.
\xnn	Matches nn, where nn is a hexadecimal number, like \x1b
\onn	Matches nn, where nn is a octal number, like \0033

Some group examples for pattern:

"[abc]"	An a, b, or c
"a|b|c"	An a, b, or c
"	Not a, b, c, but some other character

[^abc]"	
"[A-F0-9a-f]"	a hexadecimal number (0 to 9, or a word character A-F, or a-f).
"[-A]"	a Minus or an A
"[\dA-Fa-f]"	another hexadecimal number

Combinations:

\d+a number	at least one digit
\w+a word	a sequence of word characters, digits, and _.
.*	some character sequence
.+dito	with at least one character
^a.*r\.$	A sentence starting with a and ending with r and a point.
[A-Z][a-z]*	A normal word starting with an uppercase character and followed with any number of lowercase characters.
^\w+\s+(\w+)\s	The second word of a sentence.

Special sequences:

(?b)	Binary sort, A-Z does not enclose Umlaute and lowercase characters.
(?t)	Text sort, [A-B] encloses Ä, and other apostrophe A's (Á, À, Â, Å ..) , as well as lowercase characters.
(?bi)	Binary sort, ignores case: automatically enclosure of uppercase and lowercase characters.

```
n = reMatch(sexp, pattern, h[])
```

When h[] is a **Hash String** the first occurrence of the search pattern is placed in h[1].

When h[] is a **Hash Int** the location of the first occurrence of pattern is placed in h[1] and the length of the found substring in h[2].

```
n = reMatch(sexp, pattern, V:i%(0))
```

When the third parameter is an array of 32-bit integers (Dim i%(1)), then the start position of the substring is placed in i%(0) and the length in i%(1).

```
n = reMatch(array$(), pattern, from, to [, hi[]])
```

Searches pattern in the string array elements array$(from) to array$(to). The index of the first array element that contains the searched pattern is returned.

However, when the **Hash Int** variable is used as fifth parameter, the indices of all elements that contain the pattern are added to the **Hash** list. This works like VB's Filter function.

Example

Find a hexadecimal value

```
Debug.Show
Dim s$ = "zz 2a"
Trace reMatch(s$, "[A-F0-9a-f]+")
Dim hi As Hash Int
Trace reMatch(s$, "[A-F0-9a-f]+", hi[])
Debug.Print "hi[]-Found "; hi[% 1]; hi[% 2],
  Mid(s$, hi[% 1], hi[% 2])
Local hs As Hash String
Trace reMatch(s$, "[A-F0-9a-f]+", hs[])
```

```
Debug.Print "hs[]-Found", hs[% 1]
Dim ii(1) As Int
Trace reMatch(s$, "[A-F0-9a-f]+", V:ii(0))
Debug.Print "ii()-Found "; ii(0); ii(1), Mid(s$,
  ii(0), ii(1))
```

Locate in an array

```
Debug.Show
Dim a$() : Array a$() = "zz" #10 "zzz 3a " #10 "c
  = 0xaa"
Dim i As Int, hi As Hash Int
Trace reMatch(a$(), "[A-F0-9a-f]+", 0, _maxInt,
  hi[])
Debug.Print "a$()-Found at indices:"
For i = 1 To hi[%]
  Debug hi[% i]
Next i
```

Remarks

The syntax of the regular expression patterns is strongly linked to Perl's re. GFA-BASIC 32 does not support the more exotic possibilities of Perl, like {n,m}, (?#), and *?. In contrast with Perl GFA-BASIC 32 allows 8-bits ANSI characters.

The internal handling of search patterns is simpler as in Perl, the performance is a little better as well.

The **preMatch** function converts *pattern* into an internal format for faster execution. This allows for more efficient use of regular expressions in loops

See Also

preMatch, reSub, reStop, Hash

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

reSub Function

Purpose

Replaces a specified substring with another substring in a string using regular expressions.

Syntax

n = **reSub**(strvar, pattern, subst [, max])

reSub strvar, pattern, subst [, max]

n = **reSub**(array$(), pattern, subst, from, to [, hi[] [, max]])

reSub array$(), pattern, subst, from, to [, hi[] [, max]]

strvar:string variable
pattern, subst:string expression
from, to, max:iexp
hi[]:Hash Int
array$():string array
n:iexp

Description

reSub replaces occurrences of the regular expression *pattern* in the string variable strvar or string array array$() with *subst. max* specifies the maximum number of replacements to make. For example, the following statement replaces all spaces in a string:

```
reSub a$, " ", ""
```

If **reSub** is used as a function, then the return value indicates the number of replacements made. The number of spaces replaced in the following example is 3:

```
a$ = "This is a test"
n% = reSub(a$, " ", "")
```

The pattern "\s+" searches all spaces, tabs, #13, and #10.

```
a$ = "abc def   ghi"
reSub a$, "\s+", "-"  // result: "abc-def-ghi"
```

Like **reMatch**, **reSub** can replace in a one dimensional string array. The first parameter is a string array, and the second and third specify the search pattern and replacement text (like a simple **reSub)**. These parameters are followed by the smallest and largest array indices to process. When only these parameters are specified (*from*, *to*), then only the first string at *array$*(*from*) is enclosed in the search and replace. When used as a function, **reSub** returns this index. An additional *max* limits the number of replacements (bug, which see). The following replacement affects only *sa*(0), the *to* parameter is redundant.

```
Debug.Show
Dim i As Integer
Dim sa$()
Array sa$() = "Turbo PasCall" #10 "MS C" #10 "MS
  Cpp" _
  #10 "Visual Basic" #10 "Unix Perl" #10 "MS
    CSharp"
reSub sa(), " ", "-", 1, 1
For i = 0 To Dim?(sa()) - 1 : Trace sa(i) : Next
```

To replace a range of array elements a sixth parameter of type Hash Int is mandatory. (Dim hi As Hash Int). Then all string array elements between *from* and *to* are processed and their index is placed in hi[]. The return value or **reSub** is the number of replaced string elements (the same as hi[%]). When the search string pattern isn't found, then the Hash will not contain any entries, e.g. hi[%] = 0. Is hi[] omitted, then the return value is $8000000 = **_MinInt**. Not 0, because the searched string may have index = 0, with Dim x$(-9 .. 9) the index can even be negative.

Note: The parameter *max* seems to be erroneous. (see Known Issues)

Example

```
Debug.Show
Dim i As Integer
Dim sa$(), hi As Hash Int
Array sa$() = "Turbo PasCall" #10 "MS C" #10 "Cpp"
  _
  #10 "Visual Basic" #10 "Unix Perl" #10 "MS
    CSharp"
Trace reSub(sa(), " ", "-", 0, UBound(sa()), hi[])
Debug "The elements that are processed:"
For Each i In hi[]
  Debug i
Next
Debug "The results:"
For i = 0 To Dim?(sa()) - 1
  Trace sa(i)
Next
```

This prints in the Output window:

TRACE:(1):reSub(sa(), " ", "-", 0, UBound(sa()), hi[]) = 5

The elements that are processed:

```
0
1
3
4
5
```

The results:

```
TRACE:(2):sa(i) = Turbo-PasCall
TRACE:(2):sa(i) = MS-C
TRACE:(2):sa(i) = Cpp
TRACE:(2):sa(i) = Visual-Basic
TRACE:(2):sa(i) = Unix-Perl
TRACE:(2):sa(i) = MS-CSharp
```

Remarks

The VB function a$ = *Replace$*(*sexp*, *find*, *replace* [, *start*[, *count*[, *compare*]]]) is easily converted to GFA-BASIC 32.

VB Replace Function

```
Dim sexp As String
sexp = Replace(sexp, "aa", "xx")
Print sexp
```

GFA-BASIC 32 **reSub** Command

```
Dim sexp As String
' Dim Cmp = Mode(Compare)
' Mode Compare 0
reSub sexp, "aa", "xx"
' Mode Compare Cmp
Print sexp
```

For an overview of the regular expressions in pattern see reMatch.

Known Issues

1. Both the **reSub** function and command can NOT take a non-variable string as their first parameter otherwise a 'Variable?' error will be raised; instead, the string needs to be defined as a variable first, then put into the **reSub** statement as below:

```
Local a$ = "This is a test"
reSub a$, " ", "-"
```

...instead of...

```
reSub "This is a test", " ", "-"
```

---

2. There have been reports that the *max* parameter does not work as described: apparently it does limit the number of replacements, but returns an incorrect result. However, these reports may be historic as the error examples that were listed now return the correct result.

See Also

Hash, preMatch, reMatch, reStop, Replace

{Created by Sjouke Hamstra; Last updated: 08/08/2019 by James Gaite}

Collection Object

Purpose

A **Collection** object is an ordered set of items that can be referred to as a unit.

Syntax

Collection

Description

The **Collection** object provides a convenient way to refer to a related group of items as a single object. The items, or members, in a collection need only be related by the fact that they exist in the collection. Members of a collection don't have to share the same data type, because they are converted to a Variant.

An instance of a collection can be created using the **New** keyword. For example:

```
Dim X As New Collection
```

Once a collection is created, members can be added using the **Add** method and removed using the **Remove** method. Specific members can be returned from the collection using the **Item** method, while the entire collection can be iterated using the **For Each...Next** statement.

Properties

Count	Long	Returns the number of

objects

Methods

Add item[, key] [, before][, after]

item, key, before, after: Variant

Adds a member to a Collection

item	An expression of any type to add.
key	Optional. A unique string that specifies a key string that can be used, instead of a positional index, to access a member of the collection.
before	Optional. An expression that specifies a relative position in the collection. The member to be added is placed in the collection before the member identified by the ***before*** argument. If a numeric expression, ***before*** must be a number from 1 to the value of the collection's **Count** property. If a string expression, ***before*** must correspond to the ***key*** specified when the member being referred to was added to the collection. You can specify a ***before*** position or an ***after*** position, but not both.
after	Optional. An expression that specifies a relative position in the collection. The member to be added is placed in the collection after the member identified by the ***after*** argument. If numeric, ***after*** must be a number from 1 to the value of the collection's **Count** property. If a string, ***after*** must correspond to the ***key*** specified when the member referred to was added to the collection. You can specify a ***before*** position or an ***after*** position, but not both.

Remove index*index: Variant*

Removes a member at the specified position (1 … .Count), or when index is a string expression the key.

Item(index) *index: Variant*

Returns a member at the specified position (1 … .Count), or when index is a string expression the key.

Item is the default method for a **Collection** and can be left out, e.g.

```
col.Item(1) => col(1)
```

To refer to an individual member in a collection when you know the key name, use the **!** operator syntax, as shown in the following example.

```
col.Add "String", Key := "str1"
Print col!str1
```

The **!** operator increases the performance with 30%, but is only applicable with literal keys (no variables) that start with a letter (a..z). Keys are not case sensitive.

Example

```
Dim a%
OpenW 1
Coltest()

Proc Coltest()
  Dim f As Form
  Dim col As New Collection
  col.Add Win_1, "Win1"           // a form object
  col.Add "a string", "s1"        // a string
  col.Add 1.0, Before := "Win1" // a double
  Dim v As Variant       // collection member
  For Each v In col      // show positions ...
    Print TypeName(v)   // ... their type
  Next
  Print "col(1) = "; col(1)
  Print "col(""s1"") = "; col("s1")
  Print "col!s1 = "; col!s1
  Set f = col!Win1
  Print "Caption Win_1: "; f.Caption
  col.Remove 1
  Set f = Nothing
  Set col = Nothing
EndProc
```

Remarks

An object's position in the collection can change whenever a change occurs in the collection; therefore, the position of any specific object in the collection can vary.

Whether the ***before*** or ***after*** argument is a string expression or numeric expression, it must refer to an existing member of the collection, or an error occurs.

An error also occurs if a specified ***key*** duplicates the ***key*** for an existing member of the collection.

Internally, the **Collection** type is built on the **Hash** type. **Collection** is actually a special type of Hash: **Hash Variant**. Since the **Collection** is an OLE compatible type, the keys are UNICODE strings and strings must be converted to OLE strings first. GFA-BASIC 32 doesn't call the API conversion functions, but instead uses its own, faster, conversion routines. Despite these optimizations the **Hash** is much faster than the **Collection** and can be used instead in most cases.

The ! operator increases the performance with 30%, because the key isn't converted to a UNICODE string. In this special case GFA-BASIC 32 uses a non-compatible optimization to increase member access performance.

The ! operator is useful with OCX controls as well. Items stored in collections like ListImages, Buttons, Panels, etc., can be accessed using ! as well and profit from the performance increase.

See Also

Hash

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Const Command

Purpose

Declares constants for use in place of literal values.

Syntax

[Global | Local] Const name [As type] = v [, name1 [As type] = v, ...]

name: variable name
v: aexp

Description

A constant is a named item that retains a constant value throughout the execution of a program. Constants can be used anywhere in your code in place of actual values. A constant can be a string or numeric literal, another constant, or any combination that includes arithmetic or logical operators.

Unless a **Const** is declared **Global**, it has local scope.

The default type of a constant is **Long** (32-bits integer). However, simple types are allowed as well. **Const** accept typed constants as Bool, Byte, Short, Integer, Double, Single, Large, Currency, String, Date. When a type is specified, GFA-BASIC 32 checks for a valid assignment at compile time.

Without a type specifier, the type of the constant is determined from the value. A string literal will create a string constant and a date literal a Date constant. Some types of a constant can be forced to a specific type by adding a postfix to the value. By appending a @ a Currency constant is declared, a ! forces a Single, a # forces a Double.

Example

```
Const WM_USER = 0x400   ' hex literal
Const WM_PAINT = 15     ' decimal
Const WM_CLOSE = $10    ' hex literal
Const WM_USER = 0x400, WM_PAINT = 15, WM_CLOSE =
  $10, WM_QUIT = WM_CLOSE + 2
```

Implicit types

```
Const PiQuarter = Atn(1)                ' Double
Const GFAhometown = "Mönchengladbach" ' String
```

Explicit types

```
Global Const ACur = 2@               ' Currency
Global Const AFloat = 2!             ' Single
Global Const ADouble = 2#            ' Double
Const last_changing = #12.07.1996# ' Date type
```

Remarks

A constant can be given a type also by using a normal variable postfix (?, !, @, #, $, %, &).

```
Global Const ADouble# = 2
```

See Also

Enum, Global, Local, Dim

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

Enum Command

Purpose

Declares a set of 32-bit integer constants.

Syntax

Enum v1 [=value] [, v1 [=value]]…

v1, v2:variable name
value:iexp

Description

The elements of the **Enum** type are initialized to constant values within the **Enum** statement. The assigned values can't be modified at run time and can include both positive and negative numbers.

By default, the first enumerator has a value of 0, and each successive enumerator is one larger than the value of the previous one, unless you explicitly specify a value for a particular enumerator. Enumerators needn’t have unique values.

An enumeration can be declared **Local** as well as **Global**. Without indication an **Enum** is local when used inside a subroutine. When used in the main part of the program, the enumeration has global scope.

Example

```
Enum saturday,            /* saturday = 0 by default
 */ _
 sunday = 0,             /* sunday = 0 as well */ _
 monday,                 /* monday = 1 */ _
 tuesday,                /* tuesday = 2 */ _
 wednesday, _
 thursday, _
 Friday
```

Remarks

See Also

Const

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

#nn and Chr() Function

Purpose

Return either a single or a string of character from the extended ASCII table.

Syntax

**$** = #m#$h#on

**$** = **Chr**[$](m [,$h [,&on [,%b…]]])

m	*: Decimal integer*
$h	*: hexadecimal integer*
on, &on	*: Octal integer*
%b	*: Binary integer*

Description

Both these structures return a single character or a character string, determined by the arguments passed. The Pascal-type **#** can take values in decimal, hexadecimal and octal, while the more traditional basic **Chr**() function can accept all those plus binary.

Example

```
Global a$ = "Text 1" #13#10 "Text 2" #13#10
a$ = a$ & #50#$32#o62
Print a$
```

is equivalent to:

```
Global a$ = "Text 1" & Chr(13) & Chr(10) & "Text
 2" & Chr(13, 10)
a$ = a$ & Chr(50, $32, &o62)
Print a$
```

Remarks

Print #123 causes an error as the program confuses the **#** for a stream number and gives an error. However Print #123#125 does work.

The **#** (hash) character has many other uses as well. It is used with formatting strings, file channels (see Open), and Date literals (#23.07.2000#).

See Also

Asc(), Mk1$(), Mki$(),Mkl$(), Mks$(), Mkd$()

{Created by Sjouke Hamstra; Last updated: 02/10/2017 by James Gaite}

$, & and + String Concatenation Operators

Purpose

Used to force string concatenation of two expressions.

Syntax

$ = a **$** b
$ = a **&** b
$ = c$ + d$

a, b:aexp
c$, d$:sexp
result: svar

Description

$ and **&** are synonymous and can be used with numeric, string and variant (except 'Null' values) types to concatenate two or more values into a string; **+**, when used as a string concatenator (see here for more information), works only with string and variant types.

The result of any concatenation is always a String UNLESS all values are Variant Strings, when the result is a Variant.

Example

```
Dim w$
Dim x As Variant
Dim y As Variant = " Hallo"
Dim z As String = " GFA"
w$ = 20
x = (22 + 55) * (3 - 6 / 3)
z = w$ & x $ y + z
Print z                        // Prints " 2077
 Hallo GFA"
Print VarType(w$ & x $ y + z) // Prints 255 (non-
 Variant String)
```

Remarks

When the **$** and **&** are used with numeric values to create a String (not a Variant), a leading space is added by default; to prevent this behaviour, use Mode StrSpace 0.

Care should be taken when using the **+** operator for concatenation as shown in the example below:

```
Dim w$ = 20, x As Variant = 77
Print w$ + x  // Prints 97
Print w$ & x  // Prints 2077
```

For this, and other reasons, it is advised not to use the **+** operator for string concatenation.

See Also

+, Operator Hierarchy, String Data Type

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

NEAR Operator

Purpose

Used to compare two floating-point values for approximate equality.

Syntax

? = x **NEAR** y

x, y : avar

Description

NEAR can be used to reliably test whether two floating-point variables or expressions are equal.

Calculations on IEEE floating-point format expressions are performed in an internal 64-bit temporary register, which has more bits of accuracy than are stored in single-precision or double-precision variables. This often results in an IF statement returning an error which states that the intermediate calculation is not equal to the expression being compared. For example:

```
Dim x#, y#
x = 25, y = 60.1
Debug.Print x * y        ' result = 1502.5
If 1502.5 = (x * y) Then Debug.Print "equal"
```

Running the above code will NOT print "equal". In contrast, the following method using a placeholder variable will print "equal", but is still NOT a reliable technique as a test for equality:

```
Dim z# = 25 * 60.1
If z = 1502.5 Then Debug.Print "equal"
```

Note that explicit numeric type casts (! for single precision, # for double precision) will affect the precision in which calculations are stored and printed. Whichever type casting you perform, you may still see unexpected rounding results:

```
Debug.Print 69.82! + 1     ' Single precision,
 prints 70.8199996948242
Debug.Print 69.82# + 1     ' Double precision,
 prints 70.82.
```

Most numbers in decimal (base 10) notation do NOT have an exact representation in the binary (base 2) floating-point storage format used in single-precision and double-precision data types. The IEEE format cannot exactly represent (and must round off) all numbers that are not of the form 1.x to the power of y (where x and y are base 2 numbers). The numbers that can be exactly represented are spread out over a very wide range. A high density of representable numbers is near 1.0 and -1.0, but fewer and fewer representable numbers occur as the numbers go towards 0 or infinity. These limitations often cause Basic to return floating-point results different than you might expect. In the following example not even NEAR provides a solution:

```
Debug (69.82# + 1) - (69.82! + 1) '
 3.0517577442879e-07
If (69.82! + 1) NEAR (69.82# + 1) Then Debug
 "EQUAL"
```

The NEAR comparison compare too much bits.

Only an explicit typecast makes the floating point comparison possible:

```
If (69.82! + 1) = CSng(69.82# + 1) Then Debug
  "EQUAL"
```

Remarks

In GFA-BASIC 16 the 'near' comparison was performed using the == operator. However, in GFA-BASIC 32 this operator must be replaced by **NEAR** (in GFA-BASIC 32 the == operator is equivalent to =).

See Also

=, <, >, <=, >=, !=

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Peek Functions

Purpose

Reads a value with the specified data type from an address.

Syntax

Byte = **Peek**(addr)

Card = **CPeek**(addr)

Currency = **CurPeek**(addr)

Short = **DPeek**(addr)

Double = **DblPeek**(addr)

Long = **LPeek**(addr)

Large = **Peek8**(addr)

Single = **SngPeek**(addr)

String = **Peek**$(addr, len)

String = **StrPeek**(addr, len)

String = **CharPeek**(addr)

addr : address len : length of required string

Example

```
Debug.Show
Local a$ = "1234567890123456" & Chr(0)
Local d As Double = 12345678.90, s As Single =
  12345.67
Trace Hex(Peek(V:a$))
Trace Hex(CPeek(V:a$))
Trace Hex(CurPeek(V:a$))
Trace Hex(DPeek(V:a$))
Trace Hex(DblPeek(V:a$)) // Returns 0
Trace DblPeek(V:d)
Trace Hex(LPeek(V:a$))
Trace Hex(Peek8(V:a$))
Trace Hex(SngPeek(V:a$)) // Returns 0
Trace SngPeek(V:s)
Trace Peek$(V:a$, Len(a$))
Trace StrPeek(V:a$, Len(a$))
Trace CharPeek(V:a$)
```

Remarks

The Peek functions have corresponding **{}** functions and they can be used instead.

CharPeek, like **Char{}**, is most useful when used with API functions as it reads a string from an address until the next zero byte.

```
OpenW 1
Print title(Win_1.hWnd)
Do : Sleep : Until Me Is Nothing

Function title(ByVal f As Handle) As String
  Local s As String*256
  ~GetWindowText(f, V:s, SizeOf(s))
  Return CharPeek(V:s)
EndFunc
```

See Also

xx{}

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Poke Commands

Purpose

Writes a value in the specified data type to an address.

Syntax

Poke addr, byte

CharPoke addr, string

CPoke addr, card

CurPoke addr, currency

DPoke addr, short

DblPoke addr, double

LPoke addr, long

Poke$ addr, string

Poke8 addr, large

SngPoke addr, single

StrPoke addr, string

addr:address

Description

Writes a value to an address.

CharPoke, **Poke$**, and **StrPoke** write a null-terminated string to a memory address. Note that the memory must be large enough to strore the additional null.

Example

```
Debug.Show
Local a$ = Space(16), b$ = "ABCDEFGHIJKLMNOP" &
  Chr(0)
Trace b$
Poke V:a$, Byte{V:b$} : Debug "Poke V:a$,
  Byte{V:b$} -> ";a$ : a$ = Space(16)
CharPoke V:a$, CharPeek(V:b$) : Debug "CharPoke
  V:a$, CharPeek(V:b$) -> ";a$ : a$ = Space(16)
CPoke V:a$, Card{V:b$} : Debug "CPoke V:a$,
  Card{V:b$} -> ";a$ : a$ = Space(16)
CurPoke V:a$, Cur{V:b$} : Debug "CurPoke V:a$,
  Cur{V:b$} -> ";a$ : a$ = Space(16)
DPoke V:a$, Word{V:b$} : Debug "DPoke V:a$,
  Word{V:b$} -> ";a$ : a$ = Space(16)
DblPoke V:a$, Double{V:b$} : Debug "DblPoke V:a$,
  Double{V:b$} -> ";a$ : a$ = Space(16)
LPoke V:a$, Long{V:b$} : Debug "LPoke V:a$,
  Long{V:b$} -> ";a$ : a$ = Space(16)
Poke$ V:a$, b$ : Debug "Poke$ V:a$, b$ -> ";a$ :
  a$ = Space(16)
Poke8 V:a$, Large{V:b$} : Debug "Poke8 V:a$,
  Large{V:b$} -> ";a$ : a$ = Space(16)
SngPoke V:a$, Single{V:b$} : Debug "SngPoke V:a$,
  Single{V:b$} -> ";a$ : a$ = Space(16)
StrPoke V:a$, b$ : Debug "StrPoke V:a$, b$ -> ";a$
```

Remarks

The Poke functions have equivalent **{}**= versions, which can be used instead.

See Also

xx{}=, Peek Functions

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

MemMove

Purpose

Copies a block of memory

Syntax

MemMove dst, src, count

MemMove(dst, src, count)

Description

The first parameter of **MemMove** is the address of the destination and the second one the one of the source and the third one can be a constant or, for example, the length of the source to copy.

Example

```
Local a$ = "GFA Basic", b$ = Space(9)
MemMove V:b$, V:a$, 9               // This works as
 described
Print a$, b$
a$ = "GFA Basic", b$ = Space(9)
MemMove V:b$, V:a$, Len(b$)         // This doesn't
 work this way...
Print a$, b$
Local a% = 1234, b%
MemMove V:b%, V:a%, SizeOf(a%)
Print a%, b%
```

Remarks

MemMove is equal to **BlockMove** which can be used instead.

MemCpy is extremely efficient in copying Type variables. **MemCpy** is one of the rare commands that is compiled inline when *count* is a constant.

See Also

BMove, BlockMove, MemCpy

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

MemAnd, MemOr and MemXor Commands

Purpose

Perform a logical bit-wise operation on two bit patterns in memory.

Syntax

MemAnd [(] src_addr,dst_addr,count [)]

MemOr [(] scr_addr, dst_addr, count [)]

MemXor [(] scr_addr, dst_addr, count [)]

scr_addr,dst_addr	*: address*
count	*: integer expression*

Description

Each command performs a different logical bit-wise operation on two bit patterns in memory:

- **MemAnd** performs an AND, which results in the target bits being set only when the corresponding bits are set in both source and target area.
- **MemOr** an OR, results in the target bits being set when either the source or the target bits are also set.
- **MemXor** an XOR, which results in the target bits being set when the bits are set in either the source or target but not both.

In all cases, *scr_addr* specifies the address of the source, *dst_addr* the address of the destination location, and *count* specifies the number of bytes to use at both locations. The logical And results in the target bits being set only when the corresponding bits are set in both source and target area.

Example

```
OpenW 1
Win_1.FontTransparent = True
  // Needed to print properly on Win8/10
Local a%
Global a?(15), b?(15)
a?(9) = -1, b?(9) = -1, b?(10) = -1, b?(12) = -1
  // Set initial flags
test("Before any operations:")
MemAnd V:a?(0), V:b?(0), (Dim?(a?()) + 7) >> 3
  // Only b?(9) remains set
test("After MemAnd:")
b?(9) = -1, b?(10) = -1, b?(12) = -1
  // Reset flags for MemOr
MemOr V:a?(0), V:b?(0), (Dim?(a?()) + 7) >> 3
  // b?(9), b?(10) and b?(12) remain set
@test("After MemOr:")
b?(9) = -1, b?(10) = -1, b?(12) = -1
  // Reset flags for MemXor
MemXor V:a?(0), V:b?(0), (Dim?(a?()) + 7) >> 3
  // b?(9) is reset
@test("After MemXor:")

Procedure test(txt$)
  Local i%
  Print txt$
  Print "a?() - ";
  For i% = 0 To 15 : Print Str$(a?(i%), 2, 0)` :
    Next : Print
  Print "b?() - ";
  For i% = 0 To 15 : Print Str$(b?(i%), 2, 0)` :
    Next : Print
  Print
EndProc
```

Remarks

This method of memory manipulation can be particularly handy for use with databases. For example, if a database contains variables which are used as flags to mark (-1) or not to mark (0) an attribute, these method of memory manipulation can be very helpful. Using **MemAnd** an inquiry can be made to see if the markers apply to one or both attributes, with **MemOr** to see if the markers apply to either one or both attributes, while with **MemXor** an inquiry can be made to see if the markers apply to one or the other but not both attributes..

{Created by Sjouke Hamstra; Last updated: 04/03/2017 by James Gaite}

MemBFill, MemWFill, MemLFill, MemSet and MemZero Command

Purpose

Fills a memory area with a specified value.

Syntax

MemBFill[(] addr, count, value [)]
MemWFill[(] addr, count, value [)]
MemLFill[(] addr, count, value [)]

MemSet[(] addr, count, value [)]

MemZero[(] addr, count [)]

addr	*: address*
count	*: Int32 expression*
value	*: byte, Int16 or Int32 expression*

Description

All these commands fill a memory area, starting from the address *addr*, with *count* occurences of a particular value; in the case of **MemZero**, this value is always zero, whereas with the rest it can be specified in the *value* parameter.

MemBFill, **MemSet** and **MemZero** all write the value as a byte, while **MemWFill** writes it as a 16-bit Integer and **MemLFill** as a 32-bit Integer.

With all these commands, the parameters can be enclosed in brackets or not as desired.

Example

```
Local addr% = mAlloc(200)
MemBFill addr%, 200, 2
MemSet   addr%, 200, 1
MemZero  addr%, 200
MemWFill addr%, 100, 102
MemLFill addr%, 50, 15677
```

Remarks

One use for all these commands is to fill an array, but ArrayFill is much better suited, as shown below:

```
Local a%(1 To 200), n%, t#
t# = Timer
For n% = 1 To 10000 : MemLFill V:a%(1), 200, 2 :
  Next n
Trace a%(1)
Debug "MemLFill time:" & Timer - t#
t# = Timer
For n% = 1 To 10000 : ArrayFill a%(), 2 : Next n
Debug "ArrayFill time:" & Timer - t#
Trace a%(1)
Debug.Show
```

{Created by Sjouke Hamstra; Last updated: 02/03/2017 by James Gaite}

Bswap Function

Purpose

The **Bswap**() functions change the byte order of integer values.

Syntax

i% = **Bswap**[%](x)*i, x : ivar*

i& = **Bswap**&(x)*i&: word, x : ivar*

i% = **Bswap3**(x)*i, x : ivar*

i = **Bswap8**(x)*i: Int64, x : ivar*

Description

Bswap[%] changes the order of a 32 bit-Integer; **Bswap**(0x12345678) returns 0x78563412 (0x means Hex-literal).

Bswap& changes the order of a 16 bit-Integer; **Bswap&**(iexp) = **Rol&**(iexp, 8).

Bswap3 changes the order of the lower 3 bytes. This could be useful in converting BGR color values to (Blue-Green-Red) in RGB-values.

Bswap8 changes the order of a 64 bit-integer or Large.

Example

```
Print Hex$(Bswap%(0x12345678)) // 78563412
Print Hex$(Bswap&(0x12345678)) // 7856
Print Hex$(Bswap3(0x12345678)) // 785634
Print Hex$(Bswap8(0x12345678)) // 7856341200000000
```

Remarks

Bswap and **Bswap%** are identical. **Bswap** is a shortcut for:

```
MakeLongHiLo(Rol&(LoWord(i), 8), Rol&(HiWord(i),
  8))
```

Bswap8 is a shortcut for:

```
Print MakeLargeHiLo( _
  MakeLongHiLo(Rol&(LoWord(i64), 8), Rol&
    (HiWord(i64), 8)), _
  MakeLongHiLo(Rol&(LoWord(HiLarge(i64)), 8), Rol&
    (HiWord(HiLarge(i64)), 8)) _
  )
```

Alternatively (but slower):

```
Cv8(Mirror$(Mk8(i64)))
```

See Also

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

MakeWord Functions

Action

Makes a 16-bit integer from two bytes.

Syntax

z = **MakeWord**(hi, lo)

z = **MakeWordHiLo**(hi, lo)

z = **MakeWordLoHi**(lo, hi)

hi, lo:Byte
z:Short

Description

MakeWord and **MakeWordHiLo**() create a 16-bit integer value form two unsigned 8-bit integers. The first value is placed in the high order word of the word integer.

MakeWordLoHi() creates a 16-bit integer value form two unsigned 8-bit integers. The first value is placed in the high order word of the word integer.

Example

```
Debug.Show
Trace Hex(MakeWord(1, 2), 4)       // 0201
Trace Hex(MakeWordHiLo(1, 2), 4)   // 0201
Trace Hex(MakeWordLoHi(1, 2), 4)   // 0102
```

See Also

MakeL2L(), MakeL2H(), MakeL3H(), MakeL3L(), MakeL4H(), MakeL4L(), MakeLarge(), MakeLargeHiLo(), MakeLargeLoHi(), MakeLong(), MakeLongHiLo(), MakeLongLoHi(), MakeWParam()

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Shl Function

Purpose

Shifts a bit pattern left.

Syntax

Shl(m, n)

Shl&(m, n)

Shl%(m, n)

Shl|(m, n)

Shl8(m, n)

m **Shl** n

m **Shl8** n

Shl v, n

m, n:integer expression
v:variable

Description

Shl(m, n) and **Shl%** shifts the bit pattern of a 32-bit integer expressions m, n places left (Shl = SHift Left) and, optionally, stores the new value in a variable. **Shl&**(m, n) and **Shl|**(m, n) shift the bit pattern of a 16-bit or an 8-bit integer expression m respectively, n places left. **Shl8** is used to shift a **Large** integer.

The operators **Shl** and **Shl8** perform a left shift on an integer and Large, respectively.

Shl v, n assignment shifts the value in v by n and returns the value in v. The type of the operation is determined by the type of variable v.

Example

```
Local l%, l|
Debug.Show
Trace Bin$(202, 16)           // Prints
  0000000011001010
Trace Bin$(Shl(202, 4), 16)   // Prints
  0000110010100000
l% = Shl(202, 4)
Trace l%                      // Prints 3232
Trace Bin$(202, 16)           // Prints
  0000000011001010
Trace Bin$(Shl%(202, 4), 16) // Prints
  0000110010100000
l% = Shl%(202, 4)
Trace l%                      // Prints 3232
Trace Bin$(202, 8)            // Prints 11001010
Trace Bin$(Shl|(202, 4), 8)   // Prints 10100000
l| = Shl|(202, 4)
Trace l|                      // Prints 160
```

Remarks

m**<<** n is synonymous with **Shl**(m, n) and can be used instead.

As long as the result of the shift does not exceed the given width, **Shl**(m, n) is equivalent to a multiplication of m with 2^n.

Example to shift bits

Shl(63, 2) or 63 **Shl** 2

63 it binary: 0000 0000 0000 0000 0000 0000 0011 1111

Shift left: 0000 0000 0000 0000 0000 0000 0111 1110

Shift left : 0000 0000 0000 0000 0000 0000 1111 1100

Result is 124 = 63 * 4 = 63 * 2^2

Shl(-1, 4) or -1 **Shl** 4

-1 is binary: 1111 1111 1111 1111 1111 1111 1111 1111

Shift: 1111 1111 1111 1111 1111 1111 1111 1110

Shift: 1111 1111 1111 1111 1111 1111 1111 1100

Shift: 1111 1111 1111 1111 1111 1111 1111 1000

Shift: 1111 1111 1111 1111 1111 1111 1111 0000

Result is -16 = -1 * 16 = -1 * 2^4

See Also

Shr Rol, Ror, <<, >>

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Rol Function

Purpose

Rotates a bit pattern left.

Syntax

Functions:**Rol**(m,n)

Rol|(m,n)

Rol&(m,n)

Rol%(m,n)

Rol8(m, n)

Operators:m **Rol** n

m **Rol8** n

Assignment:**Rol** ivar, n

m, n:integer expression
ivar:integer variable

Description

Rol and **Rol%** shifts the bit pattern of a 32-bit integer expressions m, n places left (Rol = ROtate Left) and "wraps around" the bits moved off the left end to the right end again. The resulting new value is, optionally, stored in a variable. **Rol&**(m, n) and **Rol|**(m, n) rotate the bit pattern of a 16-bit or an 8-bit integer expression m respectively, n places left. **Ror8** rotates a **Large** integer.

Rol and **Rol8** can be used as operators as well.

Rol ivar, n rotates the value in ivar n places and stores the value back in ivar.

Example

```
Debug.Show
Local a%, l%, v As Large
Local Int x, y
Trace Bin$(202, 16)
// prints 0000000011001010
Trace Bin$(202 Rol 4, 16)
// prints 0000110010100000
l% = 202 Rol 4
Trace l%
// prints 0000110010100000
Trace Bin(l%, 16)
x = 202, y = 4
Rol x, 4
Trace Bin(x, 16)
// prints 0000110010100000
v = Large 20222022222 Rol8 64
Trace v
// prints 20222022222
```

See Also

Sar, Shl, Shr, Ror

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Crc32 Function

Purpose

computes the Cyclic Redundancy Check checksum for a range of bytes returning a 32-bit value.

Syntax

w = **Crc32**(addr, count, [old])

w = **Crc32**(str, [old])

w, old, addr, count:iexp
str:string

Description

The function **Crc32**() calculates a cyclic redundancy checksum (32-bits value) for a block of data: *count* bytes from the address *addr*. The optional parameter *old* is to be used if you want to create a checksum for more than one block, *old* must contain the checksum for the other block.

Example

```
Local a$ = "Dies ist eine Test‟"
Dim a#(10), b#(10)
Dim b% = 923454545
Mat Set a#() = 120
Mat Set b#() = -234
Dim cha_xor% = Crc32(V:a#(0), ArraySize(a#()))
Dim ch_xor% = Crc32(V:b#(0), ArraySize(b#()),
  cha_xor%)
```

```
Print Crc32(b%) // prints 2091025660
Print Crc32(a$) // 1965147545
Print cha_xor%  // 1254148786
Print ch_xor%   // 409962355
```

Remarks

The calculation of data with **CheckXorByte**, **CheckXorShort**, **CheckXorLong** (or **CheckSumxxx**()) is very fast (up to 10 times faster than **Crc16**() or **Crc32**()).

A checksum is a form of redundancy check, a simple way to protect the integrity of data by detecting errors in data that are sent through space (telecommunications) or time (storage). It works by adding up the basic components of the data, typically the asserted bits, and storing the resulting value. Anyone can later perform the same operation on the data, compare the result to the authentic checksum, and (assuming that the sums match) conclude that the data was probably not corrupted.

See Also

CheckSumByte(), CheckSumLong(), CheckSumShort(), CheckXorByte(), CheckXorLong(), CheckXorShort(), Crc16(), Crc32()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Crc16 Function

Purpose

computes the Cyclic Redundancy Check checksum for a range of bytes returning a 16-bit value.

Syntax

w = **Crc16**(addr, count, [old])

w = **Crc16**(str, [old])

w, old:16-bit integer
addr, count:iexp
str:string

Description

The function **Crc**() calculates a cyclic redundancy checksum (16-bits value) for a block of data: *count* bytes from the address *addr*. The optional parameter *old* is to be used if you want to create a checksum for more than one block, *old* must contain the checksum for the other block.

Example

```
Local a$ = "Dies ist eine Test“"
Dim a#(10), b#(10)
Mat Set a#() = 120
Mat Set b#() = -234
Dim cha_xor& = Crc16(V:a#(0), ArraySize(a#()))
Dim ch_xor& = Crc16(V:b#(0), ArraySize(b#()),
  cha_xor&)
```

```
Print Crc16(a$) // -31370
Print cha_xor&  // 432
Print ch_xor&   // 1827
```

Remarks

The calculation of data with **CheckXorByte**, **CheckXorShort**, **CheckXorLong** (or **CheckSumxxx**()) is very fast (up to 10 times faster than **Crc16**() or **Crc32**()).

A checksum is a form of redundancy check, a simple way to protect the integrity of data by detecting errors in data that are sent through space (telecommunications) or time (storage). It works by adding up the basic components of the data, typically the asserted bits, and storing the resulting value. Anyone can later perform the same operation on the data, compare the result to the authentic checksum, and (assuming that the sums match) conclude that the data was probably not corrupted.

See Also

CheckSumByte(), CheckSumLong(), CheckSumShort(), CheckXorByte(), CheckXorLong(), CheckXorShort(), Crc16(), Crc32()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

CheckSumByte Function

Purpose

Computes the checksum for a range of bytes

Syntax

b = **CheckSumByte**(addr%, count%, [old])

b, old: byte expression
addr, count: integer expression

Description

The function **CheckSumByte**() calculates a simple checksum for a block of data: *count* bytes from the address *addr*. The optional parameter *old* is to be used if you want to create a checksum for more than one block, *old* must contain the checksum for the other block.

The checksum is a simple adding of 8-bit values (bytes) in the data.

Example

```
Local a$, b$, ch_a|
Debug.Show
a$ = "This is a test"
b$ = "another block"
ch_a| = CheckSumByte(V:a$, Len(a$))
Trace CheckSumByte(V:a$, Len(a$))          // 249
Trace CheckSumByte(V:b$, Len(b$))          //  33
Trace CheckSumByte(V:a$, Len(a$), ch_a|)   // 243
```

Remarks

The calculation of data with **CheckSumByte**, **CheckSumShort**, **CheckSumLong** (or **CheckXorxxx**()) is very fast (up to 10 times faster than **Crc16**() or **Crc32**()).

A checksum is a form of redundancy check, a simple way to protect the integrity of data by detecting errors in data that are sent through space (telecommunications) or time (storage). It works by adding up the basic components of the data, typically the asserted bits, and storing the resulting value. Anyone can later perform the same operation on the data, compare the result to the authentic checksum, and (assuming that the sums match) conclude that the data was probably not corrupted.

See Also

CheckSumByte(), CheckSumLong(), CheckSumShort(), CheckXorByte(), CheckXorLong(), CheckXorShort(), Crc16(), Crc32()

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

CheckXorByte Function

Purpose

Computes the checksum for a range of bytes returning a byte value

Syntax

b = **CheckXorByte**(addr, count, [old])

b, old:8-bit integer
addr, count:iexp

Description

The function **CheckXorByte**() calculates a simple checksum (byte value) for a block of data: *count* bytes from the address *addr*. The optional parameter *old* is to be used if you want to create a checksum for more than one block, *old* must contain the checksum for the other block.

The checksum is a simple XOR-ing of 8-bit values in the data.

Example

```
Local a$ = "This is a Test"
Print CheckXorByte(V:a$, Len(a$)) // 75
Dim a#(10), b#(10)
Mat Set a#() = 120
Mat Set b#() = -234
Dim cha_xor| = CheckXorByte(V:a#(0), ArraySize(a#
  ()))
```

```
Dim ch_xor| = CheckXorByte(V:b#(0), ArraySize(b#
  ()), cha_xor|)
Print cha_xor|, ch_xor| // 30, 243
```

Remarks

The calculation of data with **CheckXorByte**, **CheckXorShort**, **CheckXorLong** (or **CheckSumxxx**()) is very fast (up to 10 times faster than **Crc16**() or **Crc32**()).

A checksum is a form of redundancy check, a simple way to protect the integrity of data by detecting errors in data that are sent through space (telecommunications) or time (storage). It works by adding up the basic components of the data, typically the asserted bits, and storing the resulting value. Anyone can later perform the same operation on the data, compare the result to the authentic checksum, and (assuming that the sums match) conclude that the data was probably not corrupted.

See Also

CheckSumByte(), CheckSumLong(), CheckSumShort(), CheckXorByte(), CheckXorLong(), CheckXorShort(), Crc16(), Crc32()

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Crypt Function

Purpose

The **Crypt** function is used to encrypt and decrypt data.

Syntax

Crypt[$] (Key$, Data$)

Key, Data: sexp

Description

Data$ is the buffer holding the data to be encrypted. *Key*$ specifies the key (max 116 characters = 924 bits) to be used for encryption, which will be used to start the random value generator. **Crypt** uses a symmetrical coding system, which means that the same key is used to encrypt and to decrypt the data.

Example

```
OpenW 1
Global secretData As Variant, a$, key$
key$ = "GFA Software"
secretData = "GFA Software GmbH"
Print "Key: " + key$
' Encrypt:
a$ = Crypt(key$, secretData)
Print "Encoded: " + a$
' Decrypt:
secretData = Crypt(key$, a$)
Print "Decode: " + secretData
```

```
While InKey = "" : Print AT(1, 5); "Press any key
  to close." : Wend
CloseW 1
```

Remarks

Don't use a key more than once. Don't make it easy to hack. One way to use the same key is to add the length of the data to the key: **Crypt**(key$+Str(Len(dat$), dat$).

The length of the period of the internal random generator is quite long, so that cracking will take a long time.

Crypt() can be used in many situations. But for really important security issues, you should use a DES or RSA encryption, like PGP. The result of **Crypt** can be hacked with a brute attack by trying all keys and then scanning the result for readable parts. Better computers also mean better and faster ways to use brute attack possibilities. You can increase the safety by using a checksum function (**Crc16**/**Crc32**) after encrypting to validate the encrypted data.

A general weakness with encrypting is the handling of a sequence of bytes with the same value (0-bytes, or spaces). **Crypt** hides these noticeable sequences, but it is still a weakness. That is why encrypting often is performed on packed data (**Pack**/**UnPack**)

Crypt("","") returns a random key of 128 characters.

See Also

Pack, Crc16, Crc32

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Pack, UnPack Function

Purpose

Compresses a string at byte level

Syntax

$ = Pack[$](string [,flag = 0])

$ = UnPack[$](string)

string:sexp
flag:iexp

Description

The function **Pack** function returns a compressed string from a string. The function **UnPack** decompresses a string compressed with **Pack**.

Pack will place a 12 byte label in front of a compressed string. The first four signs are "PCK0" (PeCehKahZero), after this, four more signs follow with the length of the compressed data and last four with the original length:

"PCK0" + Mkl$(length_after_compression) + Mkl$(original length) + packed data

When both the original data size as the compressed data size are smaller than 65536, a header of 8 bytes is used, with a lowercase k instead of K, and both lengths in a 16-bit value. Data that cannot be compressed (random byte sequences or a Crypt$) are marked with a lowercase c, followed by only one length (k=16 bit, K=32 bit), so 6 or 8 bytes.

The optional *flag* can have a value of 0, 1, or 2. If flag = 1 an additional bit pack run is performed. This run will take a bit of time, but as a result, you get a better compression rate (1-10%, sometimes more). In addition, plain text snippets are mostly removed from the compressed string. Packing with default value of flag (= 0) often results in a compressed string where words might be readable. A packed string with flag is 1 is marked as PCK1 or PCk1 instead of PCK0.

flag = 2 forces a bit pack, whether or not the packed string becomes longer.

Example

```
OpenW 1
Local a$, b$, c$, d$, b%, x%
Local e$, f$, g$
// Write 1000 times Hello
For b% = 0 To 9999
  a$ = a$ + "Hello"
Next
// Pack with flag 0,1,2
b$ = Pack(a$, 0)
c$ = Pack(a$, 1)
d$ = Pack(a$, 2)
// Show the length of the strings
Print Len(a$), Len(b$), Len(c$), Len(d$)
// Unpack the strings
e$ = UnPack(b$)
f$ = UnPack(c$)
g$ = UnPack(d$)
Print Len(a$), Len(e$), Len(f$), Len(g$)
Print a$ = e$, a$ = f$, a$ = g$
```

Remarks

The compression rate of **Pack** compares to ARC, the grand father of all compression programs, or Compress the program from Microsoft.

See Also

PackMem

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

MiMeDecode and MiMeEncode Functions

Purpose

Encodes and decodes a MiMe based64 encoded string.

Syntax

content$ = **MiMeDecode**(string)

content$ = **MiMeEncode**(string)

Description

Creates and reverses a mime based64 encoded string back.

Example

```
OpenW 1
Local a$, s_mime$, s$
a$ = "GFA Software Technologies GmbH"
Print "Original:   "; a$ : Print
s_mime$ = MiMeEncode(a$)
Print "Encoded: "; s_mime$
s$ = MiMeDecode(s_mime$)
Print "Decoded: "; s$
```

Remarks

There are also two keywords **_MiMeEncode** and **_MiMeDecode** which seem to be unrelated but correlate with each other as do **MiMeEncode** and **MiMeDecode** BUT do not perform the same conversion with the latter being the correct versions for MiMe64. The differences can be seen better if you run the example below:

```
Debug.Show
Local a$ = "GFABasic32"
Trace MiMeEncode(a$)
Trace _MiMeEncode(a$)
Trace MiMeDecode(MiMeEncode(a$))
Trace _MiMeDecode(_MiMeEncode(a$))
Trace MiMeDecode(_MiMeEncode(a$))
Trace _MiMeDecode(MiMeEncode(a$))
```

See Also

MemToMiMe(), MemToUU(), MiMeToMem(), MiMeDecode(), MiMeEncode(), UUToMem(), UUDecode(), UUEncode()

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

MemToMiMe, MiMeToMem Functions

Purpose

Encodes a memory block into the MiMe based64 format or decodes a Mime based64 encoded string to memory.

Syntax

$ = **MemToMiMe** (addr, len)

len% = **MiMeToMem**(str$, addr%)

addr, len:iexp

Description

MemToMiMe() converts a memory block at address *addr* and with a size of *len* into MIME format and returns it in a string.

MiMeToMem() is the reverse function of **MemToMime**(). The function reverses a mime based64 encoded string and copies it to the memory pointed to by *addr*%. The return value is the length of the decoded data string.

Example

```
OpenW 1
Local a$, length%, s_mime$, x%, content$
a$ = "GFA Software Technologies GmbH"
s_mime$ = MemToMiMe(V:a$, Len(a$))
Print s_mime$
```

```
Lset a$ = ""
length% = MiMeToMem(s_mime$, V:a$)
Print length%
Print a$
content$ = MiMeDecode(s_mime$)
Print content$
```

Remarks

When a string is encoded/decoded the alternative functions **MimeEncode**() and **MimeDecode**() can be used instead.

See Also

MemToUU(), MiMeDecode(), MiMeEncode(), UUToMem(), UUDecode(), UUEncode()

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

MemToUU, UUToMem Functions

Purpose

Encodes a memory block into the UUE format or decodes a UUE encoded string to memory.

Syntax

str = **MemToUU** (addr, len)

len% = **UUToMem**(string, addr%)

addr, len:iexp

Description

MemToUU() converts a memory block at address *addr* and with a size of *len* into UUE format and returns it in a string.

UUToMem() is the reverse function of **MemToUU**(). The function reverses a UUE encoded string and copies it to the memory pointed to by *addr*%. The return value is the length of the decoded data string. When addr% = 0 the function returns the required amount of memory to store the decoded string.

Example

```
OpenW 1
Local a$, length%, s_mime$, x%, content$
a$ = "GFA Software Technologies GmbH"
```

```
s_mime$ = MemToUU(V:a$, Len(a$))
Print s_mime$
Lset a$ = ""
length% = UUToMem(s_mime$, V:a$)
Print length%
Print a$
content$ = uudecode(s_mime$)
Print content$
```

Remarks

When a string is encoded/decoded the alternative functions **UUEncode**() and **UUDecode**() can be used instead.

See Also

MemToMiMe(), MiMeToMem(), MiMeDecode(), MiMeEncode(), UUDecode(), UUEncode()

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

V: and VarPtr Functions

Purpose

Returns the address of a variable or an array element.

Syntax

% = **V:**x

% = **VarPtr**(x)

x:name of a variable of any type

Description

V: and **VarPtr** are synonymous and, in the case of strings, return the address of the string itself (not the descriptor address), in the case of arrays they return the address of an array element and in the case of simple variables the address of the variable.

Example

```
OpenW # 1
Local c%
Dim a(3) As Double
// prints the address of a(3)
Print VarPtr(a(3))
// prints the address of a%
Print VarPtr(c%)
Local b$ = "Test"
Print "Get the string using Char{addr} of b$: "; _
  Char{V:b$}
```

```
Print "Address of b$: "; V:b$
Print "Descriptor of b$: "; * b$
Print "Length of b$: "; Int{V:b$ - 4}
```

Remarks

The descriptor of a string contains the length of the string, is 4 bytes long, and is located right in front of the actual data. The descriptor address is obtained using **ArrPtr**(a$) or ***a$**.

See Also

ArrPtr, *

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Declare Command

Purpose

Declares a function in a DLL.

Syntax

Declare Function Name [**CDecl**] [**Lib** "libname"] [**Alias** "aliasname"] ([paramlist]) [**As** RetType = Long]

Declare Sub Name [**CDecl**] [**Lib** "libname"] [**Alias** "aliasname"] ([paramlist])

Declare SubA Name [**CDecl**] [**Lib** "libname"] ([paramlist])

Declare FunctionA Name [**CDecl**] [**Lib** "libname"] ([paramlist]) [**As** RetType = Long]

Declare LIB "libname"

Declare Function name **BuiltIn** "aliasname"

Description

The **Declare** command is used to declare references to external procedures or functions in a dynamic-link library. The **Declare** statement syntax has these parts:

Sub	Optional (either **Sub** or **Function** must appear). Indicates that the procedure doesn't return a value.
Function	Optional (either Sub or Function must appear). Indicates that the procedure returns

	a value that can be used in an expression.
CDecl	Optional. Required when the DLL function is a C/C++ function (see **CCall**). Default is "StdCall".
Name	Any valid procedure name. Note that DLL entry points are case sensitive.
Lib	Optional. Indicates that a DLL or code resource contains the procedure being declared. The Lib clause must be included or set before using **Declare LIB** "libname".
libname	Name of the DLL or code resource that contains the declared procedure.
Alias	Optional. Indicates that the procedure being called has another name in the DLL. This is useful when the external procedure name is the same as a keyword. You can also use Alias when a DLL procedure has the same name as a variable, constant, or any other procedure. Alias is also useful if any characters in the DLL procedure name aren't allowed by the DLL naming convention.
aliasname	Optional. Name of the procedure in the DLL or code resource. If the first character is not a number sign (#), aliasname is the name of the procedure's entry point in the DLL. If (#) is the first character, all characters that follow must indicate the ordinal number of the procedure's entry point.
paramlist	Optional. List of variables representing arguments that are passed to the procedure when it is called.
RetType	Optional. Data type of the value returned by a Function procedure; may be Byte, Boolean, Integer, Long (default), Large, Currency,

Single, Double, Date, String (variable length only), or Variant, a user-defined type, or an object type.

The *paramlist* argument has the following syntax and parts:

[**ByVal** | **ByRef**] varname [**As** type = Long]

ByVal	Optional. Indicates that the argument is passed by value. Mandatory with dynamic **String** parameters, even when they must receive a return value. (ByRef would pass the address of the string descriptor.) A fixed **String** may be passed as **ByRef**.
ByRef	Indicates that the argument is passed by reference. **ByRef** is the default.
varname	Name of the variable representing the argument being passed to the procedure; follows standard variable naming conventions. The name is informational only.
type	Data type of the argument passed to the procedure; may be Byte, Boolean, Integer, Long (default), Currency, Single, Double, Date, String (variable length only), Object, Variant, a user-defined type, or an object type. When the '**As** type' is omitted, **Long** is assumed.

The *paramlist* can not contain an array or a **ParamArray** declaration. A user-defined type is to be passed as **ByRef**. A **String** parameter has to be declared as **ByVal**.

SubA and **FunctionA** exclude the **Alias** clause; they are used to use the ANSI version of the declared DLL procedure. GFA-BASIC 32 generates the alias by itself by appending the 'A' to the DLL procedure name.

Using all default settings a DLL function can be declared as:

```
Declare LIB "version"
Declare FunctionA GetFileVersionInfo(ByVal
  Filename$, ByVal dwhandle, ByVal dwlen, ByVal
  lpData)
```

The **BuiltIn** variant doesn't seem to work.

Example

```
Declare Function GetUserName Lib "advapi32.dll"
  Alias "GetUserNameA" (ByVal lpBuffer As String,
  nSize As Long) As Long
Declare FunctionA  GetUserName Lib "advapi32.dll"
  (ByVal lpBuffer As String, nSize As Long) As Long
'
Dim uname As String = String(30, 0)
GetUserName(uname, 30)
Print ZTrim(uname)
```

Remarks

A **Declare'd** DLL function is loaded when the function is used the first time. In the background the API functions *LoadLibrary*() and *GetProcAddress*() are invoked. A missing DLL isn't noticed before the function is called, therefore.

FreeDll explicitly releases a DLL from memory. The argument filename$ should be exactly the same as the DLL name specified in the **Declare** statement. Filename$ may contain a path.

The **~** (void operator) is no longer necessary to void the return value of a DLL function. In addition, DLL functions are no longer called using @@ or ^^, but simply by their name as if they were common functions.

Note ~ is still necessary for built-in API functions.

Built in API functions

GFA-BASIC 32 supports more than 1000 API functions, functions that can be used as any other GFA-BASIC 32 function. Only the standard API functions from User, Kernel and GDI are implemented, other not often used API functions like for instance WinSock functions are to be declared explicitly.

The type of the parameters of the built-in API-Functions is not checked upon compiling. Each parameter is assumed to be a 32-bit integer. A string can be passed to an API function, but is always copied to one of the 32 internal 1030-Byte buffer BEFORE the address of the buffer is passed. See String Data type.

A user defined Type (As type) is always passed by reference, so that its address is passed (automatically **V:**).

Note - These rules don't apply to DLL functions introduced with the **Declare** statement. Here GFA-BASIC 32 behaves like VB and the rules for calling such APIs must be respected.

Some API function names are already in use by GFA-BASIC 32 and are therefore renamed. ***GetObject***() becomes ***GetGdiObject***(), ***LoadCursor*** becomes ***LoadResCursor***. Obsolete functions are not implemented, obviously.

The winapi32.inc.g32 contains the declarations that are not included in GFA-BASIC 32 itself. This file also describes VB to GB32 conversion tips. The often used "**As Any"** type clause in VB is used to declare a *void pointer*, a pointer to anything, a typeless parameter. The "**As Any"** type is not supported in GFA-BASIC 32 (with reasons) and should be replaced by **ByVal** … **As Long**. You then pass the address of the variable to the DLL function.

The \Include folder contains more declaration files, both in g32 source code format as well as compiled libraries.

See Also

FreeDll, V:, String, StdCall

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

DisAsm Object

Purpose

Disassembler for GFA-BASIC 32 code.

Syntax

Dim name **As New DisAsm**

name: variable name

Description

Like many other debug facilities of GFA-BASIC 32, the disassembler is invoked at the code level. A new instance of the disassembler object is created by using **New** with the **DisAsm** type name. The **DisAsm** method of the disassembler disassembles an instruction at a given address, specified with the **Addr** property. After disassembling the instruction, **Addr** is incremented with the number of bytes occupied by the instruction. The next time **DisAsm** is executed the next instruction is disassembled.

```
00D707E5: FF 55 B4 call dpt -76[ebp]

00D707E8: 68 C4 1F E8 00 push 15212484

00D707ED: B8 20 AA C3 00 mov eax,12823072

00D707F2: 50 push eax

00D707F3: FF 15 54 26 4D 00 scall DIMNEWOBJ
```

The first column contains the (virtual) memory address of the command. The second column shows the code bytes that are disassembled to the instruction in the third column. The second column can be omitted when you set the **ByteFlag** property to 0.

The **HexDump** property changes the output from disassembly to a hex dump. A hex dump shows the hexadecimal value of binary code and the ASCII representation. This is useful when you want to examine a piece of data memory. Here an example:

```
00D70965: FF 55 B4 68 C4 1F E8 00 ÿU´hÄ.è.

00D7096D: B8 20 AA C3 00 50 FF 15 ¸ ªÃ.Pÿ.

00D70975: 54 26 4D 00 FF 55 B4 FF T&M.ÿU´ÿ

00D7097D: 35 20 AA C3 00 6A FF 8B 5 ªÃ.jÿ‹
```

The first column contains the (virtual) memory address of the hex dump. The second column contains 8 consecutive bytes found at that address. The third column shows the ASCII representation of those bytes. The number of bytes to dump in one line can be set with the **HexDumpCount** property (here: 8, default = 16).

Properties Addr |ByteFlag |HexDump |HexDumpCount |PreferHex

Addr [= long] - Returns or sets the start address of the binary code for the next disassembly or hex dump.

ByteFlag [= Bool] - Returns or sets a value determining the display of the code bytes in a disassembly listing.

HexDump [= Bool] - Returns or sets a value determining the function of the **DisAsm** method. When True a hex dump is performed, when False (default) the **DisAsm** method displays the disassembly.

HexDumpCount [= long] - Returns or sets a value determining the number of bytes to dump in one line (default = 16).

PreferHex [= Bool] - Returns or sets a value determining the display format of addresses. If True the addresses are formatted in hexadecimal format only, and if False (0 is default) in decimal as well.

Methods DisAsm

DisAsm - Disassembles next instruction or displays the next **HexDumpCount** number of bytes as a hex dump. **DisAsm** is the default for the **DisAsm** object and can be omitted.

Example

```
20
Dim dis As New DisAsm   // a new instance of
  disassembler
dis.ByteFlag = True     // code bytes as Hex bytes
dis.HexDump = True      // disassembly or a
  HexDump
dis.HexDumpCount = 8    // bytes per line 1-32
  (16=default)
dis.PreferHex = 1       // addresses in hex format
dis.Addr = LabelAddr(20)
21
Debug.Show
While dis.Addr < LabelAddr(21)
```

```
  Debug.Print dis         // dis.DisAsm ( = default )
Wend
```

Remarks

The disassembler converts binary code into a sequence of assembly commands. Thus, for analysis of the disassembled code it is necessary to know machine commands, their binary format, and their Assembly representation. Also, it is important to understand the structure of data representation in computer memory, as well as to know the structure of programs written for the Windows operating system.

The disassembler recognizes all standard 80x86, protected, FPU, and MMX instructions.

Any disassembly lines containing

00D70B25: FF 55 B4 call dpt -76[ebp]

indicate a call to the GFA-BASIC 32 debugger. This call is generated before each statement to invoke a **Tron** procedure if it is enabled. It also allows a program to be debugged using the tray debugger. These calls are not generated when **$StepOff** is specified.

For more information on inline assembler see Asm

See Also

Asm, Debug, Tron, $Stepoff

{Created by Sjouke Hamstra; Last updated: 13/08/2019 by James Gaite}

Scaling in Forms

Description

In general, most properties of a form are stored and returned in pixels (there are one or two oddities such as Width and Height which are returned in Twips, but these are rare), with coordinates starting from zero on the x- and y-axes. However, this may not always suit how you wish to display controls and GDI objects in a form and so, in common with Visual Basic, GFABASIC32 offers the option to create a customised coordinate system, or *Scale*, so the form better suits what you, as the programmer, which to achieve.

Scaling is implemented by the changing of the *scaling factor* - the size of the standard unit of measurement - and of the starting coordinates on one or both axes. In addition, changes can be made to scaling at any point of the drawing of the form: those objects drawn before the changes keep their original scaling whilst those drawn afterwards adopt the new attributes.

Note: For form scaling to affect OCX Controls, the OcxScale property of the form must be set to True.

Changing Scaling Factors

ScaleMX, ScaleMY properties Show

ScaleWidth, ScaleHeight properties Show

ScaleMode property and ScaleMode$ function Show

Changing Starting Coordinates

ScaleLeft, ScaleTop properties Show

Changing Both

Scale method Show

ScaleMMOO function Show

Manual Scaling & Conversion

If you do not wish to permanently affect the *scaling factor* of a form, GFABASIC32 has numerous functions that allow you to perform one-off conversion between different measurement types, some of which are listed below:

ScaleX, ScaleY	These functions allow conversion between Twips, Points, Pixels, Characters, Inches, Millimetres, Centimetres, and HiMetrics; in addition, when used in a *scaled* form, they can convert between the current user-defined scaling and the standard measurements listed above.
TwipsPerHimet, HimetsPerTwips	**Screen** object properties, these allow conversion between Twips and HiMetrics.
HimetsToPixelX, HimetsToPixelY, PixelsToHimetX, PixelsToHimetY	Built-in functions that convert between HiMetrics and Pixels on both the X and Y planes.
TwipsPerPixelX, TwipsPerPixelY, PixelsToTwipX, PixelsToTwipY	The first two are **Screen** and **Form** based properties, the last two built-in functions: all allow conversion from Pixels to Twips along the specified plane or axis.
PixelsPerTwipX, PixelsPerTwipY, TwipsToPixelX, TwipsToPixelY	As with those above but converting from Twips to Pixels.

{Created by James Gaite; Last updated: 08/03/2018 by James Gaite}

Line Command

Purpose

Draws a line on the screen.

Syntax

Line x1, y1, x2, y2 [,[color] [[,**B** | **BF**]]

Line (x1, y1) - (x2, y2) [,[color] [[,**B** | **BF**]]

Line x1, y1 **To** x2, y2 [,[color] [[,**B** | **BF**]]

Line - (x2, y2) [[,color] [,[**B** | **BF**]]

Line To x2, y2[[,color] [,[**B** | **BF**]]

Line [**Step**] x1, y1, [**Step**] x2, y2 [,[color] [[,**B** | **BF**]]

x1, y1, x2, y2:Single exp
color:iexp

Description

Line x1, y1, x2, y2 draws a line on the screen from the point with coordinates x1,y1 to the point with coordinates x2,y2. The origins of the coordinate system are in the upper left corner of the screen.

Step - Optional. Keyword specifying that the starting point coordinates are relative to the current graphics position given by the **CurrentX** and **CurrentY** properties *color* - Optional. **Long** integer value indicating the RGB color used to draw the line. If omitted, the **ForeColor** property setting is used. You can use the **RGB** function or **QBColor** function to specify the color.

B - Optional. If included, causes a box to be drawn using the coordinates to specify opposite corners of the box.

F - Optional. If the B option is used, the F option specifies that the box is filled with the same color used to draw the box. You cannot use F without B. If B is used without F, the box is filled with the current **Color** and **DefFill**. The default value for F is transparent.

Example

```
OpenW # 1
Local i%
Color Rand(_C) - 1
For i% = 0 To 100 Step 2
  Line 0, 0, Rand(_X), Rand(_Y)
Next i%
```

Draws lines as rays emanating from 0,0.

Remarks

The width, style, and color of the line can be defined using **DefLine** and **Color, RGBColor, QBColor, BkColor** commands.

Line (x1, y1) - (x2, y2) ,, BF is similar to **PBox** x1,y1, x2, y2

When **Line** executes, the **CurrentX** and **CurrentY** properties are set to the end point specified by the arguments.

See Also

Draw, Color, RGBColor, QBColor, BkColor, PBox, Box

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

Box, PBox Commands

Purpose

Draws a rectangle.

Syntax

Box x1,y1,x2,y2
Box x1,y1 **To** x2,y2
Box x1,y1, **Step** w,h

PBox x1,y1,x2,y2
PBox x1,y1 **To** x2,y2
PBox x1,y1, **Step** w,h

x1,y1,x2,y2,w,h : single exp

Description

Box x1,y1,x2,y2 and **Box** x1,y1 **To** x2,y2 both draw a rectangle with the diagonally opposite corner coordinates at x1,y1 (upper left) and x2,y2 (lower right), while **Box** x1,y1 **Step** w,h also draws a rectangle but with top left coordinate x1,y1 and a width of w and height of h.

The width of the line drawn depends on the setting of the **DefLine** command, while the way a line or box is drawn on the background depends on the setting of the **DrawMode** and **BkColor** properties.

The **PBox** command acts very much the same, except that the boxes drawn are filled with a pattern defined using **Deffill**.

Example

```
OpenW 1
Box 10, 10, 100, 100
DefLine 1
Box 110, 10, Step 90, 90
PBox 10, 110, 100, 200
DefFill 5 : DefLine  0
PBox 110, 110, Step 90, 90
```

See Also

BkColor, DefFill, DefLine, DrawMode, RBox, PRBox, Box3D, PBox3D, PolyLine, PolyFill

{Created by Sjouke Hamstra; Last updated: 22/06/2017 by James Gaite}

Color Command

Purpose

Sets RGB value for the drawing and background color.

Syntax

Color f%, b%

f%, b%integer expression

Description

f% specifies the RGB color for the foreground and b% the background color to be used for drawing.

Color is the same as **ForeColor** and **BkColor**.

Example

```
Local Int n
For n = 1 To 4
  Print "GFABasic32"
  Color Rand(_C) + 1, colBackGround
Next n
```

See Also

SysCol, RGBColor, QBColor, ForeColor, BkColor

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

QBColor Function

Purpose

Sets the foreground and background color of a Form.

Syntax

QBColor fore, back

rgb = **QBColor**(index)

fore, back, index, rgb:iexp

Description

QBColor sets a color from the standard 16 VGA colors. The arguments *fore* and *back* must specify a value from 0 to 15.

The function **QBColor**(index) returns the RGB-value for the specified VGA color with *index*.

The arguments *fore*, *back* and *index* can be one of:

0 - black

1 - dark red

2 - dark green

3 - yellow-green

4 - dark blue

5 - blue-red

6 - green-grey

7 - bright grey

8 - dark grey

9 - bright red

10 - bright green

11 - bright yellow

12 - bright blue

13 - magenta

14 - turquoise

15 - white

Example

```
// draw all 16 colors in a color table
OpenW 1
Local i%, a%, b%
a% = _X / 8
For i% = 0 To 7
  QBColor i%
  PBox i% * a%, 0, (i% + 1) * a%, _Y / 2
Next i%
For i% = 8 To 15
  QBColor i%
  PBox (i% - 8) * a%, _Y / 2, i% * a%, _Y
Next i%
Do
  Sleep
Until Me Is Nothing
```

Remarks

In contrast to earlier GFABASIC versions, the **Color** command takes rgb color values rather than an index in the VGA color table. Instead of using the old Color index statement, you can now use QBColor in either of two ways.

```
Color QBColor(7)
QBColor 7
```

See Also

Color, RGBColor, SysCol

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

AutoRedraw, Image Properties

Purpose

AutoRedraw creates a persistent memory bitmap and sets the output from graphic commands to the bitmap. The **Image** property returns the bitmap as **Picture** object.

Syntax

[*Form*.]**AutoRedraw** [= *iexp*]

Set pic = Form.**Image**

pic:Picture Object

Description

Enables automatic repainting of a **Form** object. Graphics and text drawn using GFA-BASIC 32 commands are written to both the screen and to an image stored in memory. Windows API functions should use the special memory device context **hDC2** handle to draw on the memory bitmap.

AutoRedraw = 0	disable
AutoRedraw = 1	device dependent bitmap
AutoRedraw = 2	device independent bitmap (DIB)

In contrast with VB, the **Form** object does receive **Paint** events when **AutoRedraw** is enabled. The client area is repainted when necessary using the image stored in memory, but additional drawing can take place in the **Paint** event sub. The graphic output in the **Paint** event is, of course, drawn in the **AutoRedraw** bitmap as well.

The **AutoRedraw** image can be obtained using the **Image** property. The **Image** and **Picture** properties are normally used when assigning values to other properties, when saving with the **SavePicture** statement, or when placing something on the Clipboard. You can't assign these to a temporary variable, other than the **Picture** data type. There is no image when AutoRedraw = 0.

Example

```
OpenW 1
AutoRedraw = 2   // a DIB
PBox 10, 10, _X - 20, _Y - 40
Print
Print HimetsToPixelX( Me.Image.Width )
Print HimetsToPixelY( Me.Image.Height )
Do
  Sleep
Until IsNothing(Me)

Sub Win_1_Paint
  Text 0, 0, "Paint"
EndSub
```

Remarks

AutoRedraw is a property of the Form object type. Used without a Form object the current active form (**Me**) is affected.

If **AutoRedraw** = 1 or 2, there exist a _**DC2**, a memory device context. If **AutoRedraw** = 0 _**DC2** = Null.

Note **AutoRedraw** does not have a Boolean type, you should not test for **AutoRedraw** == **True**. True represents -1 and not 1 or 2, which are valid values.

See Also

Form, Picture, _DC2

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

CreateObject Function

Purpose

Creates and returns a reference to an OLE object.

Syntax

Set objectvariable **= CreateObject(**"progID", ["servername"]**)**

Description

You can use the **CreateObject** function in a **Set** statement to create a new object and assign an object reference to an object variable. You must specify the object's programmatic identifier as an argument to the function, and the object you want to access must be externally creatable.

The *progID* argument is usually the fully qualified class name of the object being created; for example, Word.Document. However, progID can be different from the class name. For example, the progID for a Microsoft Excel object is "Sheet" rather than "Worksheet." The optional *servername* argument can be specified to create an object on a remote machine across a network. It takes the Machine Name portion of a share name. For example, with a network share named \\MyServer\Public, the *servername* argument would be "MyServer."

The following code example starts Microsoft Excel (if Microsoft Excel is not already running) and establishes the variable xlApp to refer to an object of the Application class.

The argument "Excel.Application" fully qualifies Application as a class defined by Microsoft Excel:

Dim xlApp As Object

Set xlApp = CreateObject("Excel.Application")

Example

```
' Declare an object variable to hold the object
' reference. Dim as Object causes late binding.
Dim ExcelSheet As Object
Set ExcelSheet = CreateObject("Excel.Sheet")
' Make Excel visible through the Application
  object.
ExcelSheet.Application.Visible = True
' Place some text in the first cell of the sheet.
ExcelSheet.Worksheets("Sheet1").Range("A1").Value
  = "This is column A, row 1"
' Save the sheet to test.xls in the application
  directory.
ExcelSheet.SaveAs App.Path & "\TEST.xls"
' Close Excel with the Quit method on the
  Application object.
ExcelSheet.Application.Quit
' Release the object variable.
Set ExcelSheet = Nothing
' Tidy up line
Kill App.Path & "\TEST.xls"
```

This code starts the application creating the object, in this case, a Microsoft Excel spreadsheet. Once an object is created, you reference it in code using the object variable you defined. Then, you access properties and methods of the new object using the object variable, *ExcelSheet*, and other Microsoft Excel objects, including the *Application* object and the *Cells* collection.

For those who do not have Microsoft Excel, the following example invokes an instance of the ubiquitous Internet Explorer:

```
// Dim a generic object variable
Dim ie As Object
// Assign a new occurence of Internet Explorer to
  the object
Set ie =
  CreateObject("InternetExplorer.Application")
// Use IE's in-built APIs to manipulate the object
ie.navigate
  "http://www.gfabasic32.blogspot.co.uk/"
ie.visible = True
// Create an alternative means of closing Internet
  Explorer
OpenW Center  1, , , 130, 90 : Win_1.ControlBox =
  False
Ocx Command cmd = "Close IE", 10, 10, 100, 22
Do : Sleep : Until Win_1 Is Nothing

Sub cmd_Click
  Try
    ie.quit
  Catch
    // RPC Error - Internet Explorer already closed
  EndCatch
  CloseW 1
  Set ie = Nothing
EndSub
```

Remarks

See Also

Automation, GetObject

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

App Object

Purpose

The **App** object is a global COM object accessed with the **App** keyword.

Syntax

App

Description

App determines or specifies information about the application's title, version information, the path, and name of its executable file.

There can be one **App** object only. You cannot create another **App** object like:

Dim MyApp As New App

You can however assign the **App** object to a variable of type **App.** By setting another object variable to the same object, the reference count for that object is incremented. After using the object variable it should be set to **Nothing** to decrement the reference count.

```
Dim MyApp As App   ' a variable of Type App
Set MyApp = App    ' set to global App
' use it
Set MyApp = Nothing
```

There would be little use for this, though.

Properties/Methods

Arguments | AvailPageFile | AvailPhys | AvailVirtual | Comments | CompanyName | FileDescription | FileName | FileVersion | Forms | hInstance | InternalName | LegalCopyright | LegalTrademarks | Major | MajorRevision | MemoryLoad | Minor | Name | OriginalFilename | Path | PrivateBuild | ProdMajor | ProdMajorRevision | ProdMinor | ProdRevision | ProductName | ProductVersion | Revision | scArguments | scClear | scCommonPrograms | scCommonStartMenu | scDescription | scDirectory | scHotkey | scIconIndex | scIconPath | scPath | scPrograms | scRead | scShowCmd | scSpecialDir | scStartMenu | scWrite | SpecialBuild | TotalPageFile | TotalPhys | TotalVirtual | WinCompany | WinUser

Known Issues

Similar to **mAlloc**(-1) through to **mAlloc**(-4), **AvailPageFile**, **AvailPhys**, **TotalPageFile** and **TotalPhys** are currently broken in most versions of Windows after XP SP3. See the mAlloc() page for the workaround.

See Also

Screen, mAlloc()

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Screen Object

Purpose

Returns information about the desktop and other OS settings.

Syntax

Screen

Description

Screen retrieves various system metrics (widths and heights of display elements) and system configuration settings. Most of the properties conform to the *GetSystemMetrics* API function or its GFA-BASIC 32 counterpart **SysMetric**(). However, these functions return all values in pixels, where the **Screen** object returns some properties in Twips. In addition, there are quite some properties and methods (GetDC, Fonts) not found with the **SysMetric** function.

Arrange	Integer	Specifies how the system arranged minimized windows
CleanBoot	Integer	Specifies how the system was started: 0 Normal boot, 1 Fail-safe boot, 2 Fail-safe with network boot
cMetrics	Integer	Number value for the parameter

		GetSystemMetrics
cMouseButtons	Integer	Number of buttons on mouse (2,3 or 0)
cxBorder, cyBorder	Integer	Width and height, in pixels, of a window border.
cxCursor, cyCursor	Integer	Width and height, in pixels, of a cursor.
cxDlgFrame, cyDlgFrame	Integer	Thickness, in pixels, of the frame around the perimeter of a window that has a caption but is not sizable.
cxDoubleClk, cyDoubleClk	Integer	Width and height, in pixels, of the rectangle around the location of a first click in a double-click sequence. The second click must occur within this rectangle for the system to consider the two clicks a double-click.
cxDrag, cyDrag	Integer	Width and height, in pixels, of a rectangle centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins.
cxEdge, cyEdge	Integer	Dimensions, in pixels, of a 3-D border.
cxFixedFrame,	Integer	Thickness, in pixels, of

cyFixedFrame		the frame around the perimeter of a window that has a caption but is not sizable.
cxFrame, cyFrame	Integer	Thickness, in pixels, of the sizing border around the perimeter of a window that can be resized.
cxFullScreen, cyFullScreen	Integer	Width and height of the client area for a full-screen
cxHScroll, cyHScroll	Integer	Width, in pixels, of the arrow bitmap on a horizontal scroll bar; and height, in pixels, of a horizontal scroll bar.
cxHThumb, cyVThumb	Integer	Width, in pixels, of the thumb box in a horizontal and vertical scroll bar.
cxIcon, cyIcon	Integer	The default width and height, in pixels, of an icon.
cxIconSpacing, cyIconSpacing	Integer	Dimensions, in pixels, of a grid cell for items in large icon view.
cxsmIcon, cysmIcon	Integer	Recommended dimensions, in pixels, of a small icon. Small icons typically appear in window captions and in small icon view.
cxMaximized,	Integer	Default dimensions, in

cyMaximized		pixels, of a maximized top-level window on the primary display monitor.
cxMaxTrack, cyMaxTrack	Integer	Default maximum dimensions, in pixels, of a window that has a caption and sizing borders.
cxMenuCheck, cyMenuCheck	Integer	Dimensions of the default menu check-mark bitmap, in pixels.
cxMenuSize, cyMenuSize	Integer	Dimensions of menu bar buttons
cxMin, cyMin	Integer	Minimum width and height of a window, in pixels.
cxMinimized, cyMinimized	Integer	Dimensions of a minimized window, in pixels
cxMinSpacing, cyMinSpacing	Integer	Dimensions of a grid cell for a minimized window, in pixels.
cxMinTrack, cyMinTrack	Integer	Minimum tracking width and height (size) of a window, in pixels.
cxScreen, cyScreen	Integer	Width and height, in pixels, of the screen of the primary display monitor (same as **x**, **y**).
cxSize, cySize	Integer	Width and height of a button in a window's caption or title bar, in pixels.

cxSizeFrame, cySizeFrame	Integer	Thickness, in pixels, of the sizing border around the perimeter of a window that can be resized.
cxsmSize, cysmSize	Integer	Dimensions, in pixels, of small caption buttons.
cxVScroll, cyVScroll	Integer	Width, in pixels, of a vertical scroll bar; and height, in pixels, of the arrow bitmap on a vertical scroll bar.
cyCaption	Integer	The height of the standard window caption in pixels.
cyKanjiWindow	Integer	For double-byte character set versions of the system, this is the height, in pixels, of the Kanji window at the bottom of the screen.
CommCtlVersion	Double	DLL version number of CommCtl.dll
dbcsEnabled	Boolean	TRUE if User32.dll supports DBCS
DEBUG	Boolean	TRUE if the debug version of User.exe is installed
FontCount	Integer	Number of installed fonts
Fonts	String	Name of installed fonts, i = 0 to Screen.FontCount -1

GetDC, **ReleaseDC**	Integer	Returns and releases DC of desktop window. Call ReleaseDC when ready.
Height, Width	Integer	Width and height, in Twips (OCX compatible), of the screen of the primary display monitor.
HimetsPerTwip, TwipsPerHimet	Double	Conversion factor for Himets to twips.
hWnd	Integer	Desktop window handle
MenuDropAlignment	Boolean	TRUE if drop-down menus are right-aligned
MidEastEnabled	Boolean	TRUE if the system is enabled for Hebrew and Arabic languages.
MouseCursor	Object	Sets and returns a MouseCursor object.
MouseIcon	Object	Sets and returns a MouseIcon object
MousePointer	Integer	Sets and returns the mouse to use.
MousePresent	Boolean	True if mouse is present
MouseX, MouseY	Integer	Mouse screen x, y position in pixels
MouseK	Integer	Mouse button state (1 = left, 2 = right, 4 = middle)
Network	Integer	Least significant bit is set if a network is present; otherwise, it is cleared.

PenWindows	Boolean	TRUE the Microsoft Windows for Pen computing extensions are installed.
PixelsPerTwipX, PixelsPerTwipY	Double	Conversion factor for pixels to twips.
Secure	Boolean	TRUE if security is present.
ShellVersion	Double	DLL version number of Shell32.dll
ShiftKeys	Integer	Shift, Ctrl, and Alt status. Returns a bit mask meaning 0 - 0x000001 - Shift 1 - 0x000001 - Control 2 - 0x000001 - Alt 3 - 0x000001 - Caps Lock 4 - 0x000020 - Windows key left 5 - 0x000020 - Windows key right 6 - 0x000040 - Menu 8 - 0x000100 - Shift left 9 - 0x000200 - Control left 10 - 0x000400 - Alt left 12 - 0x001000 - Shift right 13 - 0x002000 - Control Right 14 - 0x004000 - Alt

		right 16 - 0x010000 - Insert active 17 - 0x020000 - Num Lock active 18 - 0x040000 - Scroll Lock active 19 - 0x080000 - Alt active 20 - 0x100000 - Windows key left active 21 - 0x200000 - Windows key right active 22 - 0x400000 - Application key active
ShowSounds	Boolean	TRUE to present information visually in situations where it would otherwise present the information only in audible form.
SlowMachine	Boolean	TRUE if the computer has a low-end (slow) processor.
SwapButton	Boolean	Buttons swapped?
TwipsPerPixelX, TwipsPerPixelY	Double	Conversion factor for twips to pixels.
WinVer	String	String containing Windows version
WorkLeft, WorkTop, WorkWidth , WorkHeight	Integer	Screen work area in pixels. (When Me = Nothing, **_X** and **_Y** return WorkWidth and

		WorkHeight respectively.)
x, y	Integer	Width and height, in pixels, of the screen of the primary display monitor (same as **cxScreen**, **cyScreen**).

Event

Screen_KeyPreview

Example

Screen.ShiftKeys

```
PrintScroll = 1
SetFont SYSTEM_FIXED_FONT
Local i%
Do
  i = Screen.ShiftKeys
  Print Bin$(i, 23); " ";
  If Btst(i, 0) Then Print "Shift ";
  If Btst(i, 1) Then Print "Control ";
  If Btst(i, 2) Then Print "Alt+ ";
  If Btst(i, 3) Then Print "CapsLock ";
  If Btst(i, 4) Then Print "LWin+ ";
  If Btst(i, 5) Then Print "RWin+ ";
  If Btst(i, 6) Then Print "Appl+ ";
  If Btst(i, 8) Then Print "LShift ";
  If Btst(i, 9) Then Print "LControl ";
  If Btst(i, 10) Then Print "LAlt ";
  If Btst(i, 12) Then Print "RShift ";
  If Btst(i, 13) Then Print "RControl ";
  If Btst(i, 14) Then Print "RAlt ";
  If Btst(i, 16) Then Print "Insert* ";
  If Btst(i, 17) Then Print "NumLock* ";
  If Btst(i, 18) Then Print "ScrollLock* ";
  If Btst(i, 19) Then Print "Alt* ";
  If Btst(i, 20) Then Print "LWin* ";
  If Btst(i, 21) Then Print "RWin* ";
  If Btst(i, 22) Then Print "Appl* ";
  Print
  DoEvents
Loop Until Me Is Nothing
```

See Also

MouseCursor, MouseIcon, MousePointer

{Created by Sjouke Hamstra; Last updated: 13/08/2019 by James Gaite}

Err Object

Purpose

Contains information about runtime errors. Accepts the **Raise** and **Clear** methods for generating and clearing run-time errors.

Syntax

Err [**.**{*property* | *method*}]

Description

The **Err** object is an intrinsic object with global scope, there is no need to create an instance of it in your code. The properties of the **Err** object are set by the generator of an error - GFA-BASIC 32, an Automation object, or the programmer.

The default property of the **Err** object is **Number**. **Err** contains an integer.

See Err$ for a list of errors and exception codes.

When a run-time error occurs, the properties of the **Err** object are filled with information that uniquely identifies the error and information that can be used to handle it. To generate a run-time error in your code, use the **Raise** method.

Properties

Number | Description | HelpContext | HelpFile | Source | LastDllError | HResult | Exception

Methods

Clear | Raise | Throw

Example

The following code shows how to handle error # 46 ("Error with object"), which is often set after an error with an automation object.

```
ObjectErr()

Sub ObjectErr()
  Dim ObjectRef As Object, Msg$
  ' Try to start non existent object
  Try
    Set ObjectRef = GetObject("MyWord.Basic")
  Catch
    If Err.Number = 46
      Msg = "There was an error attempting to open
        the Automation object!" + _
        #10 "Description: " + Err.Description + _
        #10 "HResult: " + Hex(Err.HResult) + _
        #10 "Source: " + Err.Source
      MsgBox Msg
    End If
  EndCatch
End Sub
```

The next example shows two different ways how to 'throw' custom or user-defined errors:

```
Try
  CustomError1()
```

```
Catch
  ~MsgBox("Error" & Err & ": " & Err$, 0,
    Err.Source)
EndCatch
Try
  CustomError2()
Catch
  ~MsgBox("Error" & Err.Number & ": " &
    Err.Description, 0, Err.Source)
EndCatch

Proc CustomError1()
  Err.Number = 153
  Err.Source = "Custom Error1"
  Err.Description = "Random Error"
  Err.Throw
EndProcedure

Proc CustomError2()
  Err.Raise 153, "Custom Error2", "User Defined
    Error"
EndProcedure
```

Remarks

The nature of the system **Err** object/value is one of the big changes in GFA-BASIC 32. In previous version of GFA-BASIC, the **Err** value was a global system variable of type integer. In GFA-BASIC 32, it is a COM object with a default **Number** property (a long integer). As a result, the **Err** statement behaves exactly as in GFA-BASIC 16, because it is a shortcut for **Err.Number**.

The GFA-BASIC 32 error numbers are in the range from 0-152. See Err$ for a list of errors and exception codes and strings.

See Also

Error, Err$(), Try

{Created by Sjouke Hamstra; Last updated: 11/01/2017 by James Gaite}

CommDlg Ocx

Purpose

The **CommDlg** Ocx provides a standard set of dialog boxes for operations such as opening and saving files, setting print options, and selecting colors and fonts. The control also has the ability to display help.

Syntax

CommDlg

Description

The **CommDlg** object provides an interface to the routines in the Microsoft Windows dynamic-link library Commdlg.dll.

You use the **CommDlg** object in your application by adding it to a form and setting its properties. In code a CommDlg object is created using the Ocx command or the As New clause in a Dim statement.

Ocx CommDlg cd

Dim cd As New CommDlg

The dialog displayed by the Ocx control is determined by the methods of the control. The **CommDlg** object can display the following dialogs using the specified method.

ShowOpen	Show Open Dialog Box
ShowSave	Show Save As Dialog Box
ShowColor	Show Color Dialog Box

ShowFont	Show Font Dialog Box
ShowPageSetup	Show Page Setup Dialog Box
ShowPrint	Show Print or Print Options Dialog Box
ShowHelp	Invokes the Windows Help Engine
ShowFolders	Show Browse for Folders Dialog Box

There is no way to specify where a dialog box is displayed.

Properties

CancelError | Color | Colors | DefExt | DevNames | Enabled | FileName | FileTitle | Filter | FilterIndex | Flags | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | FromPage | hDC | HelpCommand | HelpContext | HelpFile | HelpKey | IniDir | Left | Max | Min | Name | Parent | pgBottom | pgLeft | pgMinBottom | pgMinLeft | pgMinRight | pgMinRight | pgMinTop | pgScale | pgTop | PointSize | Tag | Title | Top | ToPage

Methods

AboutBox | ShowColor | ShowFolders | ShowFont | ShowHelp | ShowOpen | ShowPageSetup | ShowPrint | ShowSave

Events

OnHelp

Syntax:

Sub *CommDlg*_**OnHelp**

Occurs when the user selects the Help button on the common dialog box. The Help button is displayed if the

Flags property includes the Help button flag bit.

Example

Ocx CommDlg cd

cd.Flags = cdfSHowHelp

cd.ShowFont

Sub cd_OnHelp

MsgBox "Help clicked"

EndSub

See Also

Dlg Open, Dlg Save, Dlg Color, Dlg Font, Dlg Print

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

Printer Object

Purpose

The **Printer** object enables you to communicate with a system printer (initially the default system printer).

Syntax

Printer

Description

The **Printer** object controls the current selected printer through its properties and methods. By default, the **Printer** object controls the default system printer, but this can be changed by using the **CommDlg** methods **ShowPageSetup** and **ShowPrint**, which allows the user to select a printer, or by using the **SetPrinterByName** command.

SetPrinterByName invokes a hidden **Set Printer** = statement.

After selecting a printer with **ShowPrint**, the CommDlg object must be assigned to the **Printer** object (see example). Once the **Printer** object is setup the printer options can be adjusted using the properties and methods. However, this doesn't mean that the printer itself is initialized. To initialize the printer and select the color, font, and sizes, the print job must be started. The initialization occurs when **StartDoc** is executed or when **Lprint** is invoked. **Lprint "";** initializes the printer without moving the current output position (for an example see **PrintForm**).

Note - To both read and write the properties of an individual printer, you must first make that printer the default printer for the application.

Example 1

```
// Select a printer and use landscape mode.
OpenW 1
Ocx CommDlg cd
cd.ShowPrint              // Open Printer dialog box
Set Printer = cd          // change Printer object
Printer.StartDoc "Test" // initialize
Printer.Orientation = 1 // portrait mode
Output = Printer          // change output
Printer.StartPage
FontName = "courier new"// current output
FontSize = 72
Print "Hello"
Printer.EndPage
Printer.Orientation = 2 // landscape
Printer.StartPage
Print "Hello"
Printer.EndPage
Printer.EndDoc
Output = Win_1
```

Properties

BackColor | BkColor | CurrentX | CurrentY | DefHeight | DefLeft | DefTop | DefWidth | DeviceName | dmCollate | dmColor | dmCopies | dmPaperBin | dmPaperBinName | dmPaperLength | dmPaperSize | dmPaperSizeName | dmPaperSizeX | dmPaperSizeY | dmPaperWidth | dmQuality

| dmYRes | Duplex | DrawMode | DriverName | Font | FontCount | FontBold | FontItalic | FontName | FontSize | FontStrikethru | FontTransparent | FontUnderline | Fonts | ForeColor | hDC | Height | Left | Name | Orientation | Page | PageWidth | PageHeight | PaperWidth | PaperHeight | PrintScroll | PrintWrap | PortName | ScaleHeight | ScaleLeft | ScaleMode | ScaleTop | ScaleWidth | Tag | Top | Width | Zoom

Methods

AbortDoc | EndDoc | EndPage | NewFrame | PaintPicture | Scale | ScaleX | ScaleY | SetFont | StartPage | TextHeight | TextWidth | TwipPerPixelX | TwipPerPixelY | TwipsPerPixelX | TwipsPerPixelY | PixelsPerTwipX | PixelsPerTwipY

Events

AbortProc, AutoNewFrame

Remarks

To gather information about all the available printers on the system use the **PrinterCount**, **PrinterName**, and **PrinterInfo** properties of the **App** object.

See Also

Form, CommDlg, SetPrinterByName, PrinterCount, PrinterName, PrinterInfo

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Comments

Purpose

Program comments.

Syntax

Rem[ark] *Comment* **°** *Comment*
Code **'** *Comment*
Code **//** *Comment*
Code **(*** *Comment* [***)** *Code*]
Code **/*** *Comment* [***/** *Code*]

Description

Comments are used in programming to leave explanatory notes within the program to help programmers (and maybe yourself in the future) understand how the code works and/or what it does.

There are three types of comments that can be used in GFA-BASIC:

1. **Rem**, **Remark** - This group can only be used to remark or comment an entire line (or section of a line enclosed by colons). No syntax control is performed on this line and all subsequent characters are no longer interpreted as commands, function, or variables. These are classic BASIC commands going back many years.
2. **'**, **//** - These comments can be inserted at the beginning or in the middle of a line, and all text following is considered part of the comment. **'** is an old BASIC and **//** a C form of commenting.

3. **(*...*)**, **/*...*/** - These two types can be used to bracket comments, meaning that they can be used either at the start or in the middle of a line and may have code following them; however, unlike in other languages, these are not multi-line comments (GFA-BASIC does not have a mutli-line comment option). The first element of both these types can also be used in the same way as those in the second group. **(*...*)** comes from Pascal and AppleScript and **/*...*/** is used in C, Java and SQL.

In addition to the above types, there is the GFA specific comment marker - ° - which acts in a similar way to **Rem** and **Remark**. This comment marker can be added manually to the beginning of any line OR by blocking two or more lines in the IDE (blocking just one line comments the whole program) and pressing Ctrl-I.

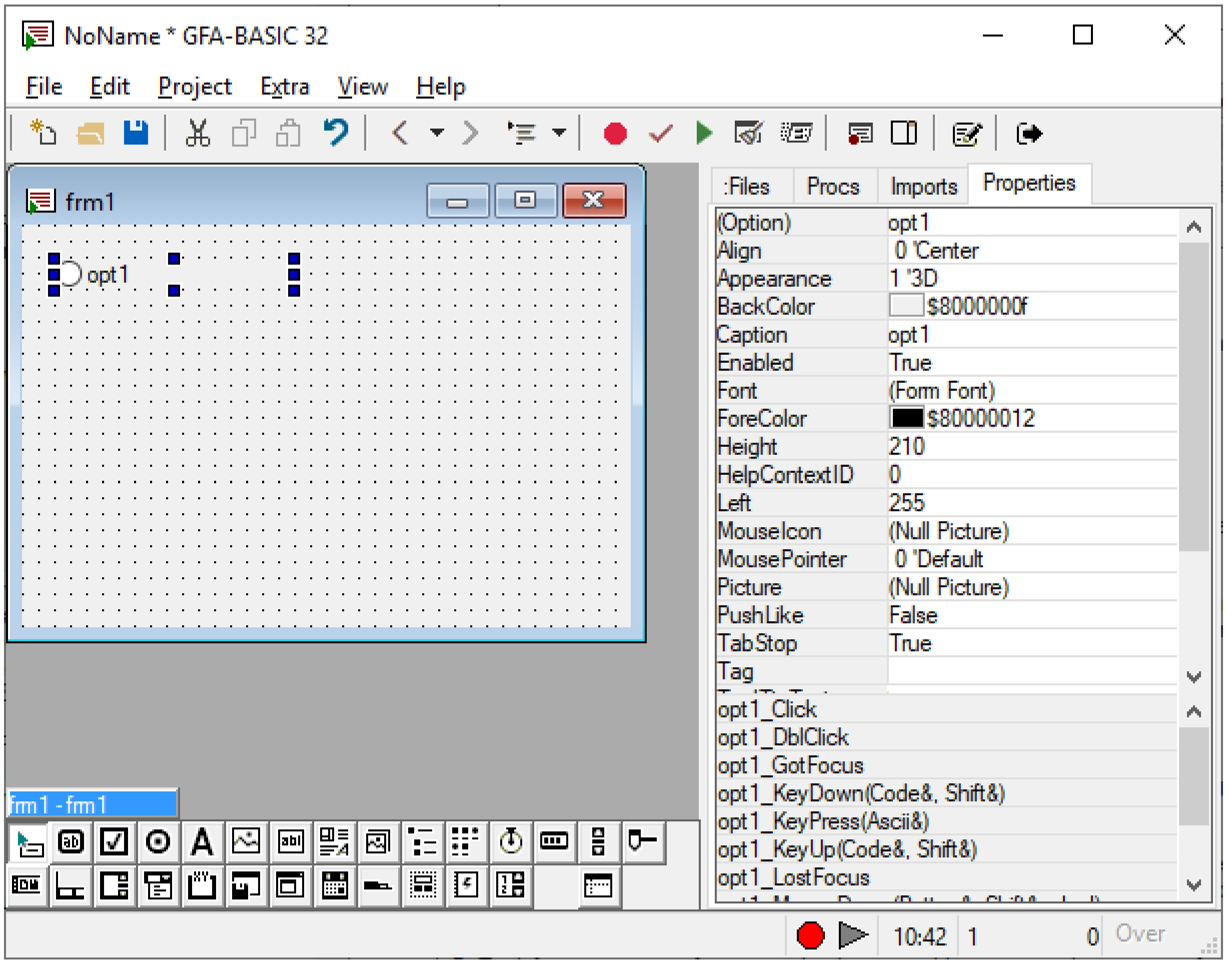
Example

```
Remark This comment takes up an entire line...
Rem    ...as does this shortened version.
Local n%    ' These comments can be used after a
  command...
n% = 1      // ...but can have not code after them.
Print (* This comment and... *) n% /* ...this one,
  can be used mid-code */ + 10
```

{Created by Sjouke Hamstra; Last updated: 27/01/2016 by James Gaite}

Setting OCX Properties

The next step is to set properties for the objects you've created. The Properties window provides an easy way to set properties for all objects on a form. When the OCX Properties window isn't visible, choose the Properties command from the View menu, or use the context menu for the control.



(Option)	opt1
Align	0 'Center
Appearance	1 '3D
BackColor	▭ $8000000f
Caption	opt1
Enabled	True
Font	(Form Font)
ForeColor	■ $80000012
Height	210
HelpContextID	0
Left	255
MouseIcon	(Null Picture)
MousePointer	0 'Default
Picture	(Null Picture)
PushLike	False
TabStop	True
Tag	

```
opt1_Click
opt1_DblClick
opt1_GotFocus
opt1_KeyDown(Code&, Shift&)
opt1_KeyPress(Ascii&)
opt1_KeyUp(Code&, Shift&)
opt1_LostFocus
```

The Properties window consists of two parts:

Properties list - The left column displays all of the properties for the selected object. You can edit and view settings in the right column. See also Using OCX Controls

Events list - The lower part displays all event subs for the OCX object. After changing the (Name) property the names of the event subs are adjusted as well. Any implemented event sub is displayed in bold. Double clicking an event will add the procedure header at the end of the source code.

Mnemonic key

Normally, keyboard users move the input focus from one control to another in a form or dialog box with the TAB and ARROW keys. However, you can define a mnemonic key that allows users to choose a control by pressing a single key. (All the mnemonics within a form/dialog box should be unique.)

To define a mnemonic key for a control with a visible caption (pushbuttons, check boxes, and radio buttons) select the control and in the Properties window in the **Caption** box, type an ampersand (&) in front of the letter you want as the mnemonic for that control. An underline appears in the displayed caption to indicate the mnemonic key.

To define a mnemonic for a control without a visible caption make a caption for the control by using a static text control. In the static text caption, type an ampersand (&) in front of the letter you want as the mnemonic. Make sure the static text control immediately precedes the control it labels in the tab order.

Next

Using OCX Controls

See Also

The Files tab, The Procs tab, The Imports tab

{Created by Sjouke Hamstra; Last updated: 24/02/2019 by James Gaite}

Boomarks & Marks

{Created by James Gaite; Last updated: 24/02/2019 by James Gaite}

Keyboard Accelerators

Moving the cursor

The editing in the GFA-BASIC editor is supported by a whole range of keyboard commands. Using the cursor block results in the following:

Arrow left	move cursor one character left
Arrow right	move cursor one character right
Arrow up	move cursor one line up
Arrow down	move cursor one line down
End	move cursor to line end
Home	move cursor to the first character of the line. Press twice to place the cursor at the beginning of the line.
Pg up	scroll one page up
Pg down	scroll one page down
Ctrl + End	move cursor to end of file
Ctrl + Home	move cursor to start of file

Holding down Shift together with the keyboard shortcuts above will result in selecting a range of characters, words, and lines.

The cursor can also be moved by using the mouse. To do this, move the mouse pointer to the desired position and click the left mouse button once.

Unnamed bookmarks

Shift + Ctrl + Up	Set unnamed bookmark

Shift + Ctrl + Down	Set unnamed bookmark
Ctrl + Arrow Up	move cursor up to line with bookmark
Ctrl + Arrow Down	move cursor down to line with next bookmark
Mouse	click with left mouse button in the margin in the editor to set or remove a boomark.

Control Keys

A range of editor functions can be invoked by pressing together the control key (Ctrl,^) and a character, without having to open the corresponding menu first. Furthermore, you can invoke editor functions which are not implemented in the menus as follows:

Ctrl + Y	deletes the line with the cursor.
Ctrl + U	performs an "undelete line". The last deleted line (Ctrl + Y) is inserted back into the text at the current position. After a Ctrl+U the recovered line remains in the internal buffer. This means that after a Ctrl+Y the function Ctrl+U can be invoked repeatedly to perform a primitive copy of the deleted line.
Ctrl + P	deletes the remainder of the current line from cursor position.
Ctrl + O	inserts the last portion of a line previously deleted with Ctrl+P at the current cursor position.
Ctrl + N	inserts a blank line above the line with the cursor.

Ctrl + R	Replace text. Use Ctrl+F3 to replace the next occurrence of the text. Use Ctrl + Shift + F3 to replace the previous occurrence.
Ctrl + A	Selects all text
Ctrl + C	Copy selection to clipboard. (Ctrl + Delete)
Ctrl + X	Cut selection to clipboard. (Shift + Delete)
Ctrl + V	Paste clipboard contents (Shift + Insert)
Ctrl + F	Search text. Use F3 to find the next occurrence of the text. Use Shift + F3 to find the previous occurrence.
Ctrl + Z	Undo (Ctrl + Backspace).
Ctrl + K	Invokes the set bookmarks context menu.
Ctrl + Q	Invokes the bookmarks selection context menu.
Ctrl + G	invokes a Dialog box for entry or a line number. The cursor then jumps to this line.
Ctrl + T	Transpose characters (swaps the character at the left with the character at the right).

Function keys

F1	Keyword Help
Shift + F1	Index Help
Alt + F2	New .chm version of the Help File (this required the GfaNewHelpAF2.gll extension to be added to the IDE)
F3	Next Find
Shift + F3	Previous Find
Ctrl + F3	Next Replace
Ctrl + Shift + F3	Previous Replace
F4	Next line containing a syntax error

Shift + F4	Previous line containing a syntax error
F5	Compile and Run
Shift + F5	Compile only (Test). This performs a full compile to test whether all loops, subroutines, and conditional statements of the program are complete. It also collects all variables and checks their correct use.
F6	Cycle through the sidebar tabs from left to right. Activates the sidebar when it isn't visible.
Shift + F6	Cycle through the sidebar tabs from right to left. Activates the sidebar when it isn't visible.
Ctrl + F6	1. Cycle through the sidebar and code editor. Activates the sidebar when it isn't visible. 2. Cycle through forms in the Form Editor.
Shift + Ctrl + F6	In the Form Editor cycles backwards through the forms.
F7	Toggle between the code and form editor. Activates the sidebar when it isn't visible.
Shift + F7	Creates a new form in the form editor. Activates the sidebar when it isn't visible.
F10	Activate the menu bar
F11	Toggle folding of current procedure.
F12	Toggle folding of current procedure and all below.

Alt keys

These key combinations are a shortcut for some of the menu items.

Alt + 0	Switch to Code Editor (F7)

Alt + 1	Switch to Form Editor (F7)
Alt + 2	Enable OCX Properties sidebar
Alt + 3	Toggle Debug Output Window
Alt + 4	Split window
Alt + 5	OCX Overview for tab order (TreeView overview of all Forms)
Alt + Backspace	Undo (Ctrl +Z)
Alt + Return	Properties dialog box.
Alt + Ctrl + R	Record keys, or end recording when recording has started.
Alt + Ctrl + P	Play recorded keyboard macro, or pause recording when recording has started.

Other keys

Insert	Toggle between insert and overwrite mode
Delete	delete the character under cursor and move the remainder of the line left.
Tab	Indent the selected text one tab stop (8 spaces) to the right. In overwrite mode Tab moves the cursor in multiples of 8 to the end of the line. TabWhen sidebar has the focus,resets the focus to the editor window.
Ctrl + Tab	Jump in multiples of 8 to the beginning of the line.
Ctrl + Break	Break program (Stop)
Ctrl + Shift + Break	Break and continue using the debugger
Enter	When in the sidebar jumps to the first

line of the selected procedure, or inserts the text of the Import element.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

LG32 Libraries

Description

When writing programs, there are user-defined commands and functions which may be used again and again in different projects and, rather than having to copy and paste the code into each new project you create that will use it (or having to update all projects using that code when revisions/bug fixes are made), it may be possible to put the commands and functions in a Library which can be called from each program that requires them.

Creating a Library Show

Loading a Library Show

Forms in a Library Show

Forms in a Library using APIs Show

Restrictions & Known Issues Show

{Created by Sjouke Hamstra; Last updated: 04/04/2018 by James Gaite}

Creating a Control

Select a tool from the Toolbox. Then click on the position in the form where you want the element to be placed. The element is placed with a default size.

Sizing and moving

A small rectangular box called *sizing handles* appears at the corners of the control; you'll use these sizing handles to resize the control. You can also use the mouse, keyboard, and menu commands to move controls, lock and unlock control positions, and adjust their positions.

To resize a control select the control you intend to resize by clicking it with the mouse. After the sizing handles have appeared on the control, position the mouse pointer on a sizing handle, and drag it until the control is the size you choose.

To move a control use the mouse to drag the control to a new location on the form. Or, use the Properties window to change the **Top** and **Left** properties.

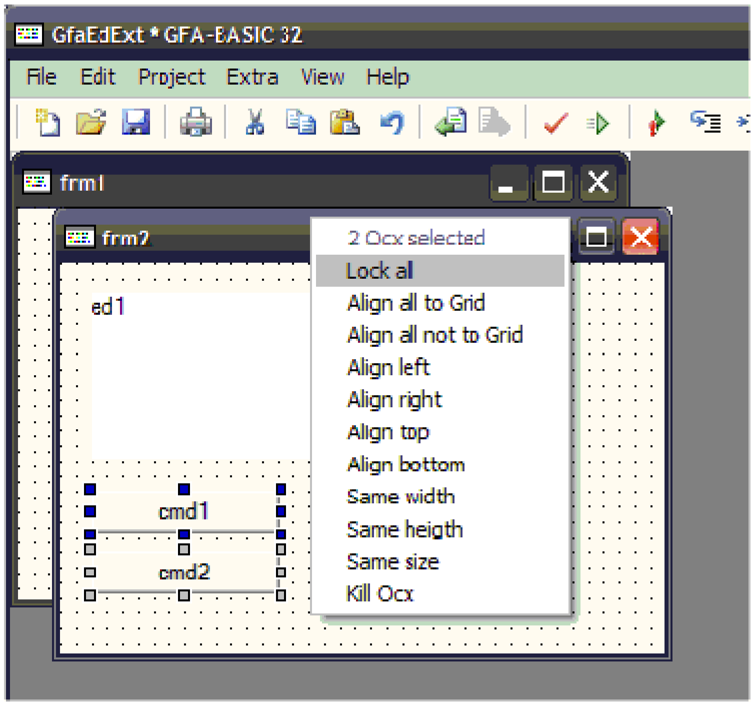
You can also use the keyboard;

Arrow keys	Move OCX or Form one grid unit a time (one grid unit is 8 pixels, or 120 or 96 twips).
Shift + Arrow keys	Resize OCX or Form one pixel a time (one pixel is 15 or 12 twips).
Ctrl +Shift + Arrow keys	Resize OCX or Form one grid unit a time.

Aligning controls

The Form editor provides layout tools that align and size controls automatically. Many layout commands are available only when more than one control is selected. When a control is selected, it has a shaded border around it with solid (active) or hollow (inactive) “sizing handles,” small squares that appear in the selection border. When multiple controls are selected, the dominant control has solid sizing handles; all the other selected controls have hollow sizing handles. You select multiple controls by holding down SHIFT and then clicking the controls. When you are sizing or aligning multiple controls, the Form editor uses the “dominant control” to determine how the other controls are sized or aligned. By default, the dominant control is the last control selected, but you can change it.

After selecting right-click on the dominant control to display the context menu with the tools to automatically size and align the controls.



From the context menu, choose one of the **Align**-options:

Align all to Grid	aligns the selected to the grid and will with new movements.
Align all not to Grid	removes grid-alignment flag from the selected controls.
Align left	aligns the selected controls along their left side.
Align right	aligns the selected controls along their right side.
Align top	aligns the selected controls along their top edges.
Align bottom	aligns the selected controls along their bottom edges.

You can resize a group of controls based on the size of the dominant control. To make controls the same width, height, or size select the controls you want to resize. Make sure the correct dominant control is selected. The final size of the controls in the group depends on the size of the dominant control. From the context menu choose one of the following commands:

Same width	Makes them the same width as the dominant control.
Same height	Makes them same height.
Same size	Makes the same size as the dominant control.

Locking

To lock all control positions, choose **Lock all** from the context menu. This will lock all selected controls on the form in their current positions so that you don't inadvertently move them once you have them in the desired location. This will lock controls only on the selected form; controls on other forms are untouched. This is a toggle command, so you can also use it to unlock control positions. To adjust the position of locked controls, you can change the control's **Top** and **Left** properties in the Property window.

Tab order

Use the 'Ocx Overview' dialog box (View menu) to set the tab order of the controls on the form. GFA-BASIC 32 determines the tab order by the order the controls are placed on the form (there is no *TabIndex* property). The tab order can be changed by dragging the controls in the 'Ocx Overview' dialog box. If you want to prevent users from tabbing to a particular control, you can set the **TabStop** property to **False** for that control, but only in code.

Next:Setting OCX Properties

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Command Ocx

Purpose

Creates an **Ocx Command** control in the current active form, window, or dialog.

Syntax

Ocx Command name = text$ [, id], x, y, b, h [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

Use a **Command** button control to begin, interrupt, or end a process. When chosen, a **Command** button appears pushed in and so is sometimes called a push button.

To display text on a **Command** button control, set its **Caption** property. A user can always choose a **Command** button by clicking it. To allow the user to choose it by pressing ENTER, set the **Default** property to **True**. To allow the user to choose the button by pressing ESC, set the **Cancel** property of the **Command** button to **True**.

Properties

Align | Appearance | Cancel | Caption | Default | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | Height, Width | HelpContextID |

hWnd | Index | Left | MouseCursor | MouseIcon | MousePointer | Name | Picture | PushLike | TabStop | Tag | Text | ToolTiptext | Top | Value | Visible | WhatsThisHelpID | Width | WinStyle

Methods

DoClick | Move | Refresh | SetFocus | SetFont | TextHeight | TextWidth | ZOrder

Events

Click | DblClick | GotFocus | LostFocus | KeyDown | KeyUp | KeyPress | MouseDown | MouseUp | MouseMove

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

Option Ocx

Purpose

Creates an **Ocx Option** control in the current active form, window, or dialog.

Syntax

Ocx Option name = text$ [, id], x, y, b, h [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

An **Option** control displays an option that can be turned on or off.

Usually, **Option** controls are used in an option group to display options from which the user selects only one. You group **Option** controls by drawing them inside a container such as a **Frame** control, or a form. To group **Option** controls in a **Frame**, draw the **Frame** first, and then draw the **Option** controls inside. All **Option** controls within the same container act as a single group.

The **Option** Ocx control has the following properties, methods, and events.

Properties

Align | Appearance | Caption | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | Height | HelpContextID | hWnd | Index | Left | Top | MouseCursor | MouseIcon | MousePointer | Name | Picture | PushLike | TabStop | Tag | Text | ToolTiptext | Value | Visible | WhatsThisHelpID | Width | WinStyle

Methods

DoClick | Move | Refresh | SetFocus | SetFont | TextHeight | TextWidth | ZOrder

Events

Click | DblClick | GotFocus | LostFocus | KeyDown | KeyUp | KeyPress | MouseDown | MouseUp | MouseMove

Remarks

CheckBox and **Option** controls function similarly but with an important difference: Any number of **CheckBox** controls on a form can be selected at the same time. In contrast, only one **Option** in a group can be selected at any given time.

To display text next to the **CheckBox**, set the **Caption** property. Use the **Value** property to determine the state of the control-selected, cleared, or unavailable.

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

CheckBox Ocx

Purpose

Creates an **Ocx CheckBox** control in the current active form, window, or dialog.

Syntax

Ocx CheckBox name = text$ [, id], x, y, b, h [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

A **CheckBox** control displays an X when selected; the X disappears when the **CheckBox** is cleared. Use this control to give the user a True/False or Yes/No option. You can use **CheckBox** controls in groups to display multiple choices from which the user can select one or more. You can also set the value of a **CheckBox** programmatically with the **Value** property.

The **CheckBox** Ocx control has the following properties, methods, and events.

Properties

Align | Appearance | Caption | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | Height | HelpContextID | hWnd | Index | Left |

Top | MouseCursor | MouseIcon | MousePointer | Name | Picture | PushLike | TabStop | Tag | Text | ThreeState | ToolTiptext | Value | Visible | WhatsThisHelpID | Width | WinStyle

Methods

DoClick | Move | Refresh | SetFocus | SetFont | TextHeight | TextWidth | ZOrder

Events

Click | DblClick | GotFocus | LostFocus | KeyDown | KeyUp | KeyPress | MouseDown | MouseUp | MouseMove

Remarks

CheckBox and **OptionButton** controls function similarly but with an important difference: Any number of **CheckBox** controls on a form can be selected at the same time. In contrast, only one **OptionButton** in a group can be selected at any given time.

To display text next to the **CheckBox**, set the **Caption** property. Use the **Value** property to determine the state of the control-selected, cleared, or unavailable.

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

Label Ocx

Purpose

Creates an **Ocx Label** control in the current active form, window, or dialog.

Syntax

Ocx Label name = text$ [, id], x, y, b, h [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

A **Label** control is a graphical control you can use to display text that a user can't change directly.

You can write code that changes the text displayed by a **Label** control in response to events at run time. For example, if your application takes a few minutes to commit a change, you can display a processing-status message in a **Label**. You can also use a **Label** to identify a control, such as a **TextBox** control, that doesn't have its own **Caption** property.

You can define a character in the **Caption** property of the **Label** as an access key. When you define an access key in a **Label** control, the user can press and hold down ALT+ the character you designate to move the focus to the next control in the tab order.

Properties

Alignment | Appearance | BackColor | BorderStyle | Caption | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | ForeColor | Height | HelpContextID | hWnd | Index | Left | MouseCursor | MouseIcon | MousePointer | MultiLine | Name | Parent | TabStop | Tag | Top | ToolTiptext | Transparent | Visible | WhatsThisHelpID | Width

Methods

HitTest | Move | Refresh | SetFont | TextHeight | TextWidth | ZOrder

Events

Click | MouseDown | MouseUp | MouseMove

Example

```
OpenW Hidden 1
With Win_1
  .ScaleMode =   basTwips
  .BackColor  = colBtnFace
  .Caption    =   "Label & TextBox"
  .Height     =   3900
  .Left       =   60
  .Top        =   345
  .Width      =   4150
EndWith
Win_1.Show
OcxScale = 1
Ocx Label lb1 = "Lbl&1:", 360, 90, 2000, 375
Ocx TextBox Text1 = "Text1", 360, 480, 3135, 375
Ocx Label lb2 = "Lbl&2:", 360, 900, 2000, 375
Ocx TextBox Text2 = "Text2", 360, 1200, 3135, 375
Ocx Label lb3 = "Lbl&3:", 360, 1700, 2000, 375
Ocx TextBox Text3 = "Text3", 360, 2040, 3135, 375
Ocx Command cmdClear = "&Clear Fields", 360, 2880,
  1455, 375
.Default          =   True
Ocx Command cmdQuit = "&Quit", 2160, 2880, 1095,
  375
.Cancel           =   True
Text1.SetFocus
Do
  Sleep
Until Me Is Nothing

Sub cmdQuit_Click
  PostMessage Win_1.hWnd, WM_CLOSE, 0, 0
End Sub

Sub cmdClear_Click
  ClearTextboxes(cmdClear.Parent)
End Sub

Sub ClearTextboxes(frm As Form)
  Local EditField As Control
  For Each EditField In frm.Controls
    If TypeOf(EditField) Is TextBox Then
      EditField.Text = ""
    End If
  Next
End Sub
```

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

Image Ocx

Purpose

Creates an **Ocx Image** control in the current active form, window, or dialog.

Syntax

Ocx Image name = text$ [, id], x, y, b, h [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

An **Image** control can display a graphic. An **Image** control can display a graphic from a bitmap, icon, or metafile, as well as enhanced metafile, JPEG, or GIF files.

The **Image** control uses fewer system resources and repaints faster than a **Form** Ocx control, but it supports only a subset of the **Form** properties, events, and methods. Use the **Stretch** property to determine whether the graphic is scaled to fit the control or vice versa.

Properties

Appearance | AutoSize | BackColor | BorderStyle | Enabled | Height | HelpContextID | hWnd | Index | Left | MouseCursor | MouseIcon | MousePointer | Name | Parent | Picture | Stretch | TabStop | TabStripIndex | Tag | Tile | Top

| ToolTiptext | Transparent | Visible | WhatsThisHelpID | Width

Methods

Move | Refresh | SetFocus | ZOrder

Events

Click | GotFocus | LostFocus | KeyDown | Keyup | KeyPress | MouseDown | MouseUp | MouseMove

Example

```
Local h As Handle, p As Picture
OpenW Hidden 1, , , 450, 500 : AutoRedraw = 1
BitBlt Screen.GetDC, 0, 0, 400, 400, Win_1.hDC2,
  0, 0, SRCCOPY
Set Me = Win_1
Get 0, 0, 399, 399, h
Set p = CreatePicture(h, False)
Cls
Win_1.Show
Ocx Label lbl = "Partial Screenshot:", 10, 10,
  100, 15
Ocx Image img = "", 10, 30, 400, 400 : Set
  img.Picture = p
Do : Sleep : Until Win_1 Is Nothing
```

Remarks

An **Image** control can act as a container and can be used in the **OcxOcx** command.

Note - GFA-BASIC 32 does not provide the ***PictureBox*** control as an image container. Instead, the **Form** Ocx is extended with a Picture property to act as a replacement.

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

TextBox Ocx

Purpose

Creates an **Ocx TextBox** control in the current active form, window, or dialog.

Syntax

Ocx TextBox name [= text$] [, id] [, x, y, b, h] [, style%]

text$	*: control text*
id%	*: control identifier*
x, y, b, h	*: integer expression*
style%	*: the control styles*

Description

A **TextBox** control, sometimes called an edit field or edit control, displays information entered at design time, entered by the user, or assigned to the control in code at run time.

To display multiple lines of text in a **TextBox** control, set the **MultiLine** property to **True**. If a multiple-line **TextBox** doesn't have a horizontal scroll bar, text wraps automatically even when the **TextBox** is resized. To customize the scroll bar combination on a **TextBox**, set the **ScrollBars** property.

Scroll bars will always appear on the TextBox when its **MultiLine** property is set to **True**, and its **ScrollBars** property is set to anything except **None** (0).

If you set the **MultiLine** property to **True**, you can use the **Alignment** property to set the alignment of text within the **TextBox**. The text is left-justified by default. If the **MultiLine** property is **False**, setting the **Alignment** property has no effect.

Properties

Alignment | Appearance | BackColor | BorderStyle | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | ForeColor | Height | HelpContextID | HideSelection | hWnd | Index | Left | MaxLength | MouseCursor | MouseIcon | MousePointer | MultiLine | Name | Parent | PassWordChar | ReadOnly | ScrollBars | SelLength | SelStart | SelText | TabStop | Tag | Text | Top | ToolTiptext | Visible | WantSpecial | WhatsThisHelpID | Width

Methods

DoClick | CharFromLine | ColFromChar | GetLineFromChar | LineCount | LineFromChar | Move | Refresh | RowFromChar | SetFont | Scroll | ScrollCaret | TextHeight | TextWidth | ZOrder

Events

Change | Click | DblClick | GotFocus | LostFocus | KeyDown, Keyup | KeyPress | MouseDown | MouseUp | MouseMove | SelChange

Example

```
OpenW Hidden 1
With Win_1
  .ScaleMode =   basTwips
  .BackColor  = colBtnFace
  .Caption    =   "Label & TextBox"
  .Height     =   3950
  .Left       =   60
  .Top        =   345
  .Width      =   4000
EndWith
Win_1.Show
OcxScale = 1
Ocx Label lb1 = "Lbl&1:", 360, 90, 2000, 375
Ocx TextBox Text1 = "Text1", 360, 480, 3135, 375
Ocx Label lb2 = "Lbl&2:", 360, 900, 2000, 375
Ocx TextBox Text2 = "Text2", 360, 1200, 3135, 375
Ocx Label lb3 = "Lbl&3:", 360, 1700, 2000, 375
Ocx TextBox Text3 = "Text3", 360, 2040, 3135, 375
Ocx Command cmdClear = "&Clear Fields", 360, 2880,
  1455, 375
.Default          =   True
Ocx Command cmdQuit = "&Quit", 2160, 2880, 1095,
  375
.Cancel           =   True
Text1.SetFocus
Do
  Sleep
Until Me Is Nothing

Sub cmdQuit_Click
  PostMessage Win_1.hWnd, WM_CLOSE, 0, 0
End Sub

Sub cmdClear_Click
  ClearTextboxes(cmdClear.Parent)
End Sub

Sub ClearTextboxes(frm As Form)
  Local EditField As Control
  For Each EditField In frm.Controls
```

```
    If TypeOf(EditField) Is TextBox Then
      EditField.Text = ""
    End If
  Next
End Sub
```

Remarks

OCX Textboxes come with certain control key combinations as default. These are:

Ctrl-C	Copy
Ctrl-H	Backspace
Ctrl-I	Tab
Ctrl-J & Ctrl-M	Carriage Return and Line Feed
Ctrl-V	Paste
Ctrl-X	Cut
Ctrl-Z	Undo/Redo
Ctrl-Delete	Deletes to the end of the line
Ctrl-End	Bottom of the box
Ctrl-Home	Top of the box

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

RichEdit Ocx

Purpose

Creates an **Ocx RichEdit** control in the current active form, window, or dialog.

Syntax

Ocx RichEdit name = text$ [, id], x, y, b, h [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

The **RichEdit** control allows the user to enter and edit text while also providing more advanced formatting features than the conventional **TextBox** control.

The **RichEdit** control provides a number of properties you can use to apply formatting to any portion of text within the control. To change the formatting of text, it must first be selected. Only selected text can be assigned character and paragraph formatting. Using these properties, you can make text bold or italic, change the color, and create superscripts and subscripts. You can also adjust paragraph formatting by setting both left and right indents, as well as hanging indents.

The **RichEdit** control opens and saves files in both the RTF format and regular ASCII text format. You can use methods of the control (**LoadFile** and **SaveFile**) to directly read and write files, or use properties of the control such as **SelRTF** and **TextRTF** in conjunction with GFA-BASIC 32's file input/output statements.

To print all or part of the text in a **RichEdit** control use the **SelPrint** method.

The **RichEdit** control supports almost all of the properties, events, and methods used with the standard **TextBox** control, such as **MaxLength**, **MultiLine**, **ScrollBars**, **SelLength**, **SelStart**, and **SelText**. Applications that already use **TextBox** controls can easily be adapted to make use of **RichEdit** controls.

Note: With **TextBox** - in this instance, given the name **tb** - it is possible to get the text by using the shortcut *text$* = **tb**; similarly, it is possible to manipulate and/or check the text using the functions **Len**(), **Left**(), **Right**(), etc. This does not work with a **RichEdit** control as the value returned contains all the RTF formatting as well and may cause an error. To use the actual unformatted text, the **Text** and **TextLength** properties should be used instead.

Properties

Appearance | BackColor | BorderStyle | BulletIndent | CharFormat | DefCharFormat | DisableNoScroll | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | ForeColor | FormatDC | FormatWidth | Height | HelpContextID | HideSelection | hWnd | Index | Left | Locked | MaxLength | MouseCursor | MouseIcon | MousePointer | MultiLine | Name | ParaFormat | Parent | ReadOnly | ScrollBars | SelAlignment | SelBold | SelBullet | SelCharOffset | SelColor | SelFontName | SelFontSize | SelHangingIndent | SelIndent | SelItalic | SelLength | SelProtected | SelRightIndent | SelRTF | SelStart | SelStrikeout | SelTabCount | SelTabs | SelText | SelUnderLine | TabStop | Tag | Text | TextLength | TextRTF | Top | ToolTiptext | Visible | WantSpecial | WhatsThisHelpID | Width

Methods

DoClick | CharFromLine | ColFromChar | Find | GetLineFromChar | LineCount | LineFromChar | LoadFile | Move | Refresh | RowFromChar | SaveFile | SelLine | SelPrint | SelPrintRect | SetFont | Scroll | ScrollCaret | Span | TextHeight | TextWidth | UpTo | ZOrder

SelLine method selects the current line and returns the line number.

Events

Change | Click | DblClick | GotFocus | LostFocus | KeyDown, Keyup | KeyPress | MouseDown | MouseUp | MouseMove | Protected | SelChange

Example

```
Ocx RichEdit rtf = "", 10, 10, 300, 200 : .BorderStyle = 3
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelItalic = 1
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelBold = 1
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelItalic = 0
rtf.SelText = String( 5, "GFA-BASIC 32 ")
Ocx RichEdit rtf_copy = "", 320, 10, 300, 200 :
  .BorderStyle = 3
Do : Sleep : Until Me Is Nothing
```

Shortcut Keys

The **RichEdit** control comes with in-built shortcut keys which are listed on the Microsoft website and a list of which are copied below. It should be noted that not all keyboard configurations will support all shortcuts, as it should also be noted that, on some keyboards, the shortcut keys may be different due to different key layout (for example, on some keyboards, Ctrl+# is used for acute accents and Ctrl+' for grave, rather than Ctrl-' and Ctrl-` as noted below).

Keys	**Operations**	**Comments**
Shift+Backspace	Generate a LRM/LRM on a bidi keyboard	BiDi specific
Ctrl+Tab	Tab	
Ctrl+Clear	Select all	
Ctrl+Number Pad 5	Select all	
Ctrl+A	Select all	
Ctrl+E	Center alignment	
Ctrl+J	Justify alignment	
Ctrl+R	Right alignment	
Ctrl+L	Left alignment	
Ctrl+C	Copy	

Ctrl+V	Paste	
Ctrl+X	Cut	
Ctrl+Z	Undo	
Ctrl+Y	Redo	
Ctrl+'+' (Ctrl+Shift+'=')	Superscript	
Ctrl+'='	Subscript	
Ctrl+1	Line spacing = 1 line.	
Ctrl+2	Line spacing = 2 lines.	
Ctrl+5	Line spacing = 1.5 lines.	
Ctrl+' (apostrophe)	Accent acute	After pressing the short cut key, press the appropriate letter (for example a, e, or u). This applies to English, French, German, Italian, and Spanish keyboards only.
Ctrl+` (grave)	Accent grave	See Ctrl+' comments.
Ctrl+~ (tilde)	Accent tilde	See Ctrl+' comments.
Ctrl+; (semicolon)	Accent umlaut	See Ctrl+' comments.
Ctrl+Shift+6	Accent caret (circumflex)	See Ctrl+' comments.
Ctrl+, (comma)	Accent cedilla	See Ctrl+' comments.
Ctrl+Shift+' (apostrophe)	Activate smart quotes	
Backspace	If text is protected, beep and do not delete it. Otherwise, delete previous character.	

Ctrl+Backspace	Delete previous word. This generates a VK_F16 code.	
F16	Same as Backspace.	
Ctrl+Insert	Copy	
Shift+Insert	Paste	
Insert	Overwrite	DBCS does not overwrite.
Ctrl+Left Arrow	Move cursor one word to the left.	On bidi keyboard, this depends on the direction of the text.
Ctrl+Right Arrow	Move cursor one word to the right.	See Ctrl+Left Arrow comments.
Ctrl+Left Shift	Left alignment	In BiDi documents, this is for left-to-right reading order.
Ctrl+Right Shift	Right alignment	In BiDi documents, this is for right-to-left reading order.
Ctrl+Up Arrow	Move to the line above.	
Ctrl+Down Arrow	Move to the line below.	
Ctrl+Home	Move to the beginning of the document.	
Ctrl+End	Move to the end of the document.	
Ctrl+Page Up	Move one page up.	If in SystemEditMode and Single Line control, do nothing.
Ctrl+Page Down	Move one page down.	See Ctrl+Page Up comments.

Ctrl+Delete	Delete the next word or selected characters.	
Shift+Delete	Cut the selected characters.	
Esc	Stop drag-drop.	While doing a drag-drop of text.
Alt+Esc	Change the active application.	
Alt+X	Converts the Unicode hexadecimal value preceding the insertion point to the corresponding Unicode character.	
Alt+Shift+X	Converts the Unicode character preceding the insertion point to the corresponding Unicode hexadecimal value.	
Alt+0xxx (Number Pad)	Inserts Unicode values if xxx is greater than 255. When xxx is less than 256, ASCI range text is inserted based on the current keyboard.	Must enter decimal values.
Alt+Shift+Ctrl+F12	Hex to Unicode.	In case Alt+X is already taken for another use.
Alt+Shift+Ctrl+F11	Selected text will be output to the debugger window and saved to %temp%\DumpFontInfo.txt.	For Debug only (need to set Flag=8 in Win.ini)
Ctrl+Shift+A	Set all caps.	
Ctrl+Shift+L	Fiddle bullet style.	
Ctrl+Shift+Right Arrow	Increase font size.	Font size changes by 1 point in the range 4pt-11pt; by 2points for 12pt-28pt; it

		changes from 28pt -> 36pt -> 48pt -> 72pt -> 80pt; it changes by 10 points in the range 80pt - 1630pt; the maximum value is 1638.
Ctrl+Shift+Left Arrow	Decrease font size.	See Ctrl+Shift+Right Arrow comments.

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

ImageList Ocx

Purpose

Creates an **Ocx ImageList** control in the current active form, window, or dialog.

Syntax

Ocx ImageList name [= text$] [, id] [, x, y, b, h] [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

An **ImageList** control contains a collection of images of the same type and size, referred to by its index. The **ImageList** control is not meant to be used alone, but as a central repository to conveniently supply other controls with images. Specifically, the ListView, TreeView, TabStrip, and Toolbar controls use an **ImageList** control to store their images.

The **ImageList** control uses bitmap (.bmp, .dib), cursor (.cur), icon (.ico), JPEG (.jpg), metafiles (.emf, .wmf), or GIF (.gif) files in a **ListImages** collection of **ListImage** items. You can add and remove images at design time or run time.

The properties of the **ImageList** control define the size and type of the images added to the **ListImages** collection. The **ImageHeight**, **ImageWidth**, and **ColorFormat** properties set the dimensions of each image, the type of the image list, and whether to create a masked bitmap for the images. These properties are set before hand, either at design time in the 'ImageList Data' dialog box or in code. At design time the **ColorFormat** combo box forces to select a color format in combination with a mask.

A *non-masked image list* consists of a color bitmap that contains one or more images. A *masked image list* consists of two bitmaps of equal size. The first is a color bitmap that contains the images, and the second is a monochrome bitmap that contains a series of masks-one for each image in the first bitmap. When a non-masked image is drawn, it is simply copied into the target device context; that is, it is drawn over the existing background color of the device context. When a masked image is drawn, the bits of the image are combined with the bits of the mask, typically producing transparent areas in the bitmap where the background color of the target device context shows through.

The **UseMaskColor** property determines that for the next image a masked image is added to the list. You specify a color (**MaskColor**) that the system combines with the image bitmap to automatically generate the masks. Each pixel of the **MaskColor** color in the image bitmap is changed to black, and the corresponding bit in the mask is set to 1. As a result, any pixel in the image that matches the specified mask color is transparent when the image is drawn (using ListImage.**Draw** or ImageList.**Overlay**).

Images can be added one by one at design time and run time. GFA-BASIC 32 also supports the **AddPart** method that adds images from a larger bitmap strip to the **ImageList** control in one step.

Properties

BackColor | ColorFormat | Enabled | ImageHeight | ImageWidth | hImageList | Left | ListImage | ListImages | MaskColor | Name | Parent | Tag | Top | UseMaskColor

hImageList returns the handle to the underlying **ImageList** common control.

Methods

Add | AddItem | AddPart | Overlay

Events

None

Example

```
Local n As Int
OpenW 1, 30, 30, 300, 300
Cls colBtnFace
Ocx ImageList iml
iml.ImageWidth = 32
iml.ImageHeight = 32
iml.ColorFormat = 0
iml.MaskColor = colBtnFace
iml.UseMaskColor = True
iml.BackColor = colBtnFace
For n = 1 To 11 : iml.Add , "gfaicon" & n,
  CreatePicture(LoadIcon(_INSTANCE, n), False) :
  Next n
Ocx TreeView tv = "", 10, 10, 260, 240
```

```
tv.LineStyle = tvwRootLines : tv.ImageList = iml
For n = 1 To 11 : tv.AddItem , , , "GFA Icon" & n,
  "gfaicon" & n : Next n
Do : Sleep : Until Win_1 Is Nothing
```

Remarks

The operating environment identifies an **ImageList** control in an application by assigning it a handle, or **hImageList**. Many ImageList-related API functions require the **hImageList** of the active window as an argument. Because the value of this property can change while a program is running, never store the **hImageList** value in a variable.

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

TreeView Ocx

Purpose

Creates an **Ocx TreeView** control in the current active form, window, or dialog.

Syntax

Ocx TreeView name [= text$] [, id%] [, x, y, w, h] [, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

A **TreeView** control displays a hierarchical list of **Node** objects, each of which consists of a label and an optional bitmap. A **TreeView** is typically used to display the headings in a document, the entries in an index, the files and directories on a disk, or any other kind of information that might usefully be displayed as a hierarchy.

After creating a **TreeView** control, you can add, remove, arrange, and otherwise manipulate **Node** objects by setting properties and invoking methods. You can programmatically expand and collapse **Node** objects to display or hide all child nodes. Three events, the Collapse, Expand, and NodeClick event, also provide programming functionality.

You can navigate through a tree in code by retrieving a reference to **Node** objects using **Root**, **Parent**, **Child**, **FirstSibling**, **Next**, **Previous**, and **LastSibling** properties. Several styles are available which alter the appearance of the control. **Node** objects can appear in one of eight combinations of text, bitmaps, lines, and plus/minus signs.

The **TreeView** control uses the **ImageList** control, specified by the **ImageList** property, to store the bitmaps and icons that are displayed in **Node** objects. A **TreeView** control can use only one **ImageList** at a time. This means that every item in the **TreeView** control will have an equal-sized image next to it when the **TreeView** control's **Style** property is set to a style which displays images.

The **TreeView** Ocx control has the following properties, methods, and events.

Properties

Appearance | BackColor | BorderStyle | Count | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | ForeColor | Height | HelpContextID | HideSelection | hWnd | ImageList | Indentation | Index | LabelEdit, | Left | LineStyle | MouseCursor | MouseIcon | MousePointer | Name | Node | Nodes | Parent | SelectedItem | Sorted | Style | TabStop | Tag | ToolTiptext | Top | Visible | WhatsThisHelpID | Width

Methods

Add | AddItem | Clear | HitTest | Item | Move | Refresh | Remove | SetFocus | SetFont | StartLabelEdit | TextHeight | TextWidth | ZOrder

Events

AfterLabelEdit | BeforeLabelEdit | Click | Collapse | DblClick | Expand | GotFocus | KeyDown, KeyUp | KeyPress | LostFocus | MouseDown | MouseUp | MouseMove | NodeClick

Example

```
Dim node As Node
Ocx TreeView tv = "", 10, 10, 230, 200
tv.Add , , , "Painters"
tv.Nodes.Add  1, tvwChild , , "Da Vinci"
tv.Add 1, tvwChild, , "Titian"
tv.AddItem 1, tvwChild, , "Rembrandt"
Set node = tv.Nodes.Add(1, tvwChild, , "Goya")
Set node = tv.Add(1, tvwChild, "David" , "David")
tv.LineStyle = tvwRootLines
tv.Style = tvwTreelinesText
tv.Indentation = 25
tv("David").Italic = True
tv.Node(3).Bold = True
tv.Nodes(4).Underline = True
tv!David.EnsureVisible ' Expand tree to see all
  nodes.
tv.SetFocus
tv("David").Selected = 1
Do
  Sleep
Until Me Is Nothing
```

Remarks

Users can navigate through a tree using the keyboard as well. UP ARROW and DOWN ARROW keys cycle downward through all expanded **Node** objects. **Node** objects are selected from left to right, and top to bottom. At the bottom of a tree, the selection jumps back to the top of the tree, scrolling the window if necessary. RIGHT ARROW and LEFT ARROW keys also tab through expanded **Node** objects, but if the RIGHT ARROW key is pressed while an unexpanded **Node** is selected, the **Node** expands; a second press will move the selection to the next **Node**. Conversely, pressing the LEFT ARROW key while an expanded **Node** has the focus collapses the **Node**. If a user presses an ANSI key, the focus will jump to the nearest **Node** that begins with that letter. Subsequent pressings of the key will cause the selection to cycle downward through all expanded nodes that begin with that letter.

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

ListView Ocx

Purpose

Creates an **Ocx ListView** control in the current active form, window, or dialog.

Syntax

Ocx ListView name = text$ [, id%], x, y, w, h[, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

The **ListView** control displays items using one of four different views. You can arrange items into columns with or without column headings as well as display accompanying icons and text.

With a **ListView** control, you can organize list entries, called **ListItem** objects, into one of four different views: Large (standard) Icons, Small Icons, List, and Report

The **View** property determines which view the control uses to display the items in the list. You can also control whether the items in the list are sorted and how selected items appear.

The **ListView** control contains **ListItem** and **ColumnHeader** objects. A **ListItem** object defines the various characteristics of items in the **ListView** control, such as:

- A brief description of the item.

- Icons that may appear with the item, supplied by an **ImageList** control.

- Additional pieces of text, called subitems, associated with a **ListItem** object that you can display in Report view.

You can choose to display column headings in the **ListView** control using the **Add** method to add a **ColumnHeader** object to the **ColumnHeaders** collection.

The **ListView** Ocx control has the following properties, methods, and events.

Properties

Appearance | Arrange | BackColor | BorderStyle | CheckBoxes | CheckedCount | CheckedItems | ColumnHeaders | Count | DefaultWidth | Enabled | ExStyle | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | ForeColor | FullRowSelect | Grid | GridLines | Height | HelpContextID | HideSelection | hWnd | Icons | Index | LabelEdit | Left | ListItems | MouseCursor | MouseIcon | MousePointer | MultiSelect | Name | Parent | SelectedItem | SmallIcons | TabStop | Tag | TextBackColor | ToolTiptext | Top | TopIndex | View | Visible | WhatsThisHelpID | Width

Methods

Add | AddItem | Clear | GetFirstVisible | HitTest | Item, | LineItem | ListItem | Move | Refresh | Remove |

SelectedCount | SelectedItems | SetFocus | SetFont | SetGrid | SnapToGrid | Sort | StartLabelEdit | TextHeight | TextWidth | VisibleCount | ZOrder

The **VisibleCount** method returns an integer from 0 to the number of items visible in the control. An item is considered visible even if only a portion of the text is visible.

Events

AfterLabelEdit | BeforeLabelEdit | Click | ColumnClick | DblClick | GotFocus | ItemClick | KeyDown, KeyUp | KeyPress | LostFocus | MouseDown | MouseUp | MouseMove

Example

```
OpenW 1, 20, 20, 500, 500
' View property
Global Enum lvwIcon = 0, lvwSmallIcon, lvwList,
  lvwReport
' Arrange property (valid for lvwIcon,
  lvwSmallIcon)
Global Enum lvwNone = 0, lvwAutoLeft, lvwAutoTop
' LabelEdit property
Global Enum lvwAutomatic = 0, lvwManual ' ListView
Dim lis As ListItems, li As ListItem
Dim chs As ColumnHeaders, ch As ColumnHeader
Ocx ImageList iml
iml.ListImages.Add , "comp",
  CreatePicture(LoadIcon(Null, IDI_APPLICATION))
Ocx ListView lv = "", 10, 10, 230, 200
lv.View = lvwReport
lv.Icons = iml
lv.SmallIcons = iml
Set ch = lv.ColumnHeaders.Add( , "1" , "Column
  #1")
ch.Width = 2000
```

```
lv.ColumnHeaders.Add , "2", "Column #2"
lv.ColumnHeaders.Add , "3" , "Column #3"
lv.Add , , "ListItem #1", "comp"
lv.ListItems.Add , , "ListItem #2", "comp"
lv.AddItem , , "ListItem #3", "comp"
lv.AddItem , , "ListItem #4", "comp"
lv.AddItem , , "ListItem #5", "comp"
lv.AddItem , , "ListItem #6", "comp"
lv.AddItem , , "ListItem #7", "comp"
lv.GridLines = True
'lv.Grid 1 // Does not work
Do
  Sleep
Until Me Is Nothing
```

Remarks

Further control on the individual list items is performed with **ListItem** objects of the **ListItems** collection.

The gfawinx library defines the following constants:

For use with the View property: lvwIcon, lvwSmallIcon, lvwList, and lvwReport.
For use with the Arrange property: lvwNone, lvwAutoTop, and lvwAutoLeft.
For use with the LabelEdit property: lvwAutomatic, lvwManual.

See Also

ListItems, ListItem

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 13/08/2019 by James Gaite}

Timer Ocx

Purpose

Creates an **Ocx Timer** control 'in' the current active form, window, or dialog.

Syntax

Ocx Timer name [= text$] [, id%], [x, y, w, h] [, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

A **Timer** control can execute code at regular intervals by causing a Timer event to occur. The **Timer** control, invisible to the user, is useful for background processing.

The **Interval** property returns or sets the number of milliseconds between calls to a **Timer** control's Timer event. You can set a **Timer** control's **Interval** property at design time or run time.

The **Timer** control's **Enabled** property determines whether the control responds to the passage of time. Set **Enabled** to **False** to turn a **Timer** control off, and to **True** to turn it on. When a **Timer** control is enabled, its countdown always starts from the value of its **Interval** property setting.

Create a Timer event procedure to handle the situation that the time of **Interval** has passed.

Properties

Enabled | hWnd | Index | Interval | Name | Parent | Tag

Events

Timer

Syntax Events

Sub *Timer*_**Timer**

Occurs when a preset interval for a **Timer** control has elapsed. The interval's frequency is stored in the control's **Interval** property, which specifies the length of time in milliseconds.

Example

```
OpenW 1
PrintScroll = 1
Ocx Timer tmr
tmr.Interval = 1000
tmr.Enabled = True
Do
  Sleep
Until Me Is Nothing

Sub tmr_Timer
  Static counter% = 0
  counter++
  Text 0, 0, "Timer Event " & counter
EndSub
```

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

ProgressBar Ocx

Purpose

Creates an **Ocx ProgressBar** control in the current active form, window, or dialog.

Syntax

Ocx ProgressBar name [= text$] [, id] [, x, y, b, h] [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

The **ProgressBar** control shows the progress of a lengthy operation by filling a rectangle with chunks from left to right.

ProgressBar control has a range and a current position. The range represents the entire duration of the operation. The current position represents the progress the application has made toward completing the operation. The **Max** and **Min** properties set the limits of the range. The **Value** property specifies the current position within that range. Because chunks are used to fill in the control, the amount filled in only approximates the **Value** property's current setting. Based on the control's size, the **Value** property determines when to display the next chunk.

The **ProgressBar** control's **Height** and **Width** properties determine the number and size of the chunks that fill the control. The more chunks, the more accurately the control portrays an operation's progress. To increase the number of chunks displayed, decrease the control's **Height** or increase its **Width**. The **BorderStyle** property setting also affects the number and size of the chunks. To accommodate a border, the chunk size becomes smaller. **Note** that any changes made to **Width** and **Height** will be negated if the **Align** property is changed after those changes have been made.

The **Smooth** property causes the control to display a contiguous progress bar instead of a segmented bar.

You can use the **Align** property with the **ProgressBar** control to automatically position it at the top or bottom of the form (**basTop**, **basLeft**, **basRight**, **basBottom**).

The **Orientation** property sets a value (**basHorizO** or **basVertO**) that determines whether the control is oriented horizontally or vertically. The **Align** property always overrules the **Orientation** property.

Properties

Align | Appearance | BorderStyle | Enabled | Height | HelpContextID | hWnd | Index | Left | Max | Min | MouseCursor | MouseIcon | MousePointer | Name | Orientation | Parent | Smooth | TabStop | Tag | Top | ToolTiptext | Value | Visible | WhatsThisHelpID | Width

Methods

Move | Refresh | ZOrder

Events

Click | MouseDown | MouseUp | MouseMove

Example

```
Local i As Int32
OpenW Center # 1, , , 400, 200
Ocx ProgressBar pro1 = "", 10, 10, 200, 40
pro1.Max = 100
pro1.Smooth = True
DoEvents
For i = 0 To 100
  pro1.Value = i
  Pause 1
Next
MsgBox "Ready!"
CloseW 1
```

Remarks

To shrink the chunk size until the progress increments most closely match actual progress values, make the **ProgressBar** control at least 12 times wider than its height.

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

Scroll Ocx

Purpose

Creates a (flat) **Ocx Scroll** scrollbar control in the current active form, window, or dialog.

Syntax

Ocx Scroll name [= text$] [, id] [, x, y, b, h] [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

The **Scroll** control is a rectangle that contains a scroll box and has direction arrows at both ends. The scroll-bar control sends a notification message to its parent whenever the user clicks the mouse in the control. The parent is responsible for updating the scroll-box position. Scroll-bar controls can be positioned anywhere in a window and used whenever needed to provide scrolling input.

Use the **Max** and **Min** properties to set the appropriate range for the control. To specify the amount of change to report in a scroll bar, use the **LargeChange** property for clicking in the scroll bar, and the **SmallChange** property for clicking the arrows at the ends of the scroll bar. The scroll bar's **Value** property increases or decreases by the values set for the **LargeChange** and **SmallChange** properties.

You can position the scroll box at run time by setting **Value** between -32768 and 32,767, inclusive.

By setting the **Appearance** property, the scrollbar is changed to a flat scroll bar (equivalent to the VB ***FlatScrollBar*** control).

The **Align** property allows a **Scroll** control to be aligned to a side of the parent (**basTop**, **basLeft**, **basRight**, **basBottom**).

The **Orientation** property sets a value (**basHorizO** or **basVertO**) that determines whether the **Scroll** control is oriented horizontally or vertically. The **Align** property always overrules the **Orientation** property.

Properties

Align | Appearance | BorderStyle | Enabled | Height | HelpContextID | hWnd | Index | LargeChange, | Left | Max | Min | MouseCursor | MouseIcon | MousePointer | Name | Orientation | Parent | SmallChange | TabStop | Tag | Top | TrackValue | ToolTiptext | Value | Visible | WhatsThisHelpID | Width

Methods

Move | Refresh | SetFocus | ZOrder

Events

Change | Click | GotFocus | LostFocus | KeyDown | Keyup | KeyPress | MouseDown | MouseUp | MouseMove | Scroll

Example

```
OpenW Center # 1, , , 400, 200
Me.BackColor = colBtnFace
Ocx Scroll sc1 = "", 10, 10, 370, 20
Ocx ProgressBar pb1 = "", 10, 50, 370, 20
With sc1
  .Min = 0 : .Max = 600
  .LargeChange = (.Max - .Min) / 10 : .SmallChange
    = 10
End With
Do
  Sleep
Loop Until Me Is Nothing

Sub sc1_Scroll()
  pb1.Value = (sc1.TrackValue * 10 / 9) / ((sc1.Max
    - sc1.Min) / 100)
EndSub

Sub sc1_Change()
  pb1.Value = (sc1.Value * 10 / 9) / ((sc1.Max -
    sc1.Min) / 100)
EndSub
```

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

Slider Ocx

Purpose

Creates an **Ocx Slider** scrollbar control in the current active form, window, or dialog.

Syntax

Ocx Slider name [= text$] [, id] [, x, y, b, h] [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

A **Slider** control is a window containing a slider and optional tick marks. You can move the slider by dragging it, clicking the mouse to either side of the slider, or using the keyboard.

Slider controls are useful when you want to select a discrete value or a set of consecutive values in a range. For example, you could use a **Slider** to set the size of a displayed image by moving the slider to a given tick mark rather than by typing a number. To select a range of values, set the **SelectRange** property to **True**, and program the control to select a range when the SHIFT key is down.

The **Align** property allows a **Slider** control to be aligned to a side of the parent (**basTop**, **basLeft**, **basRight**, **basBottom**).

The **Orientation** property sets a value (**basHorizO** or **basVertO**) that determines whether the **Slider** control is oriented horizontally or vertically. The **Align** property always overrules the **Orientation** property.

Properties

Align | Appearance | BorderStyle | Enabled | Height | HelpContextID | hWnd | Index | LargeChange | Left | Max | Min | MouseCursor | MouseIcon | MousePointer | Name | Orientation | Parent | SelectRange | SelStart | SelLength | SmallChange | TabStop | Tag | TickFrequency | TickStyle | Top | ToolTiptext | Value | Visible | WhatsThisHelpID | Width

Methods

ClearSel | GetNumTicks | Move | Refresh | SetFocus | ZOrder

Events

Change | Click | GotFocus | LostFocus | KeyDown | Keyup | KeyPress | MouseDown | MouseUp | MouseMove | Scroll

Example

```
OpenW Center # 1, , , 400, 200
Me.BackColor = colBtnFace
Ocx Slider sli1 = "", 0, 0, 200, 40
.SelectRange = True
.SelStart = 20
.SelLength = 70
.TickStyle = 2
Do
  Sleep
Loop Until Me Is Nothing
```

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

ToolBar Ocx

Purpose

Creates an **Ocx ToolBar** control in the current active form, window, or dialog.

Syntax

Ocx ToolBar name = text$ [, id], x, y, b, h [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

A **Toolbar** control contains a collection of **Button** objects used to create a toolbar that is associated with an application.

Typically, a toolbar contains buttons that correspond to items in an application's menu, providing a graphic interface for the user to access an application's most frequently used functions and commands.

The **Toolbar** control allows you to create toolbars by adding **Button** objects to a **Buttons** collection. Each **Button** object can have optional text or an image, or both, supplied by an associated **ImageList** control. You can display an image on a button with the **Image** property, or display text with the **Caption** property, or both, for each **Button** object. At run time, you can add or remove buttons from the **Buttons** collection using the **Add** and **Remove** methods (See Remarks).

To program the **Toolbar**, add code to the ButtonClick event to respond to the selected button. You can also determine the behavior and appearance of each **Button** object using the **Style** property. For example, if four buttons are assigned the 'Button Group' style (2), only one button can be pressed at any time and at least one button is always pressed.

You can create space for other controls on the toolbar by assigning a **Button** object the 'Place Holder' style (4), then positioning a control over the placeholder. For example, to place a drop-down combo box on a toolbar, add a **Button** object with the 'Place Holder' style and size it as wide as a **ComboBox** control. Then place a **ComboBox** control on the placeholder with the **OcxOcx** command.

Usability is further enhanced by programming **ToolTipText** descriptions of each **Button** object. To display ToolTips, simply assign a value to the ToolTipText property.

The **ToolBar** Ocx control has the following properties, methods, and events.

Properties

Appearance | BorderStyle | Button | Buttons | Count | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | Height | HelpContextID | hWnd | ImageList | Left | MouseCursor | MouseIcon | MousePointer | Name | Tag | ToolTiptext | Top | Visible | WhatsThisHelpID | Width

Methods

Add | AddItem | Clear | Item | Refresh | Remove | SetFont | TextHeight | TextWidth

Events

Click | ButtonClick | ButtonDblClick | DblClick | MouseDown | MouseUp | MouseMove

Example

```
Ocx ToolBar tb
tb.Buttons.Add , , "Save"
tb.Add , , "Load"
Do : Sleep : Until Me Is Nothing

Sub tb_ButtonClick(Btn As Button)
  Select Btn.Index
  Case 1 : Message "Save selected"
  Case 2 : Message "Load selected"
  EndSelect
EndSub
```

Remarks

The **Toolbar** and **Buttons** methods **Clear** and **Remove** don't work correctly and will eventually crash GFA-BASIC 32.

The **ToolBar** Ocx control implicitly changes the origin of the scaling mode. The origin is moved with *SetViewportOrgEx* API. **ScaleHeight** is decremented with the height of the toolbar. Mouse client coordinates are relative to the new origin.

See Also

Form, Command, Option, CheckBox, RichEdit, ImageList, TreeView, ListView, Timer, Slider, Scroll, Image, Label, ProgressBar, TextBox, StatusBar, ListBox, ComboBox, Frame, CommDlg, MonthView, TabStrip, TrayIcon, Animation, UpDown

Ocx, OcxOcx, Buttons, Button

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

StatusBar Ocx

Purpose

Creates an **Ocx StatusBar** control in the current active form, window, or dialog.

Syntax

Ocx StatusBar name [= text$] [, id] [, x, y, b, h] [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

A **StatusBar** control provides a window, usually at the bottom of a parent form, through which an application can display various kinds of status data. The **StatusBar** can be divided up into a maximum of sixteen **Panel** objects that are contained in a **Panels** collection.

A **StatusBar** control consists of **Panel** objects, each of which can contain text and/or a picture. Properties to control the appearance of individual panels include **Width** and **Alignment** (of text and pictures). Additionally, you can use one of seven values of the **Style** property to automatically display common data such as date, time, and keyboard states.

At run time, the **Panel** objects can be configured to reflect different functions, depending on the state of the application. For detailed information about the properties, events, and methods of **Panel** objects, see the **Panel** Object and **Panels** Collection topics.

A **StatusBar** control typically displays information about an object being viewed on the form, the object's components, or contextual information that relates to that object's operation.

The **StatusBar** Ocx control has the following properties, methods, and events.

Properties

Appearance | BorderStyle | Count | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | Height | HelpContextID | hWnd | ImageList | Left | MouseCursor | MouseIcon | MousePointer | Name | Panel | Panels | SimpleText | Style | Tag | ToolTiptext | Top | Visible | WhatsThisHelpID | Width

Methods

Add | AddItem | Clear | Item | Refresh | Remove | SetFont | TextHeight | TextWidth

Events

Click | DblClick | MouseDown | MouseUp | MouseMove | PanelClick | PanelDblClick

Example

```
Ocx ImageList iml : .ImageHeight = 16 :
  .ImageWidth = 16
iml.Add , , CreatePicture(LoadIcon(Null,
  IDI_WARNING))
Ocx StatusBar sb : '.ImageList = iml // Not
  implemented
sb.Add , , "Scroll", 5
sb.Add , , "CAPS", 3
sb.Add , , ""
sb.Add , , "Panel..."
sb.Add , , "Warning" , , 1 // Can't show icons
Do : Sleep : Until Me Is Nothing
```

Remarks

The **StatusBar** Ocx control implicitly changes the height of the scaling mode. **ScaleHeight** is decremented with the height of the statusbar.

Known Issues

The **ImageList** property has not been implemented for this object.

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

ListBox Ocx

Purpose

Creates an **Ocx ListBox** control in the current active form, window, or dialog.

Syntax

Ocx ListBox name = [text$] [, id], x, y, w, h[, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

The control is a rectangle containing a list of strings (such as filenames) from which the user can select.

To add or delete items in a **ListBox** control, use the **AddItem** or **RemoveItem** method. Set the **List**, **ListCount**, and **ListIndex** properties to enable a user to access items in the **ListBox**.

If no item is selected, the **ListIndex** property value is -1. The first item in the list is **ListIndex** = 0, and the value of the **ListCount** property is always one more than the largest **ListIndex** value.

The **ListBox** Ocx control has the following properties, methods, and events.

Properties

Appearance | BackColor | ForeColor | BorderStyle | Columns | DisableNoScroll | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | Height, Width | HelpContextID | hWnd | Index | IntegralHeight |ItemData | Left, Top | List | ListCount | ListIndex | MouseCursor | MouseIcon | MousePointer | MultiSelect | Name | NewIndex | Parent | Scrollbars | Selected | Sorted | TabStop | Tag | Text | ToolTiptext | TopIndex | Visible | WhatsThisHelpID

The ***SelCount*** property is missing. This property is particularly useful when users can make multiple selections. There is no alternative then to use the LB_GETSELCOUNT message as shown in the example.

Methods

AddItem | Clear | Find | FindExact | FindNext | InsertItem | Move | Refresh | RemoveItem | SetFocus | SetFont | TextHeight | TextWidth | ZOrder

Events

Click | DblClick | GotFocus | LostFocus | KeyDown, KeyUp | KeyPress | MouseDown | MouseUp | MouseMove

Example

```
Form frm = "Listbox", , , 500, 400
Ocx ListBox lb1 =  "", 0, 0, 250, 200
.MultiSelect = 1
Ocx ListBox lb2 = "", 250, 0, 250, 200
Ocx Command cmd1 = "Add to 2", 100, 220, 80, 24
cmd1.Enabled = False
```

```
Dim i%
For i = 0 To Screen.FontCount - 1
  lb1.AddItem Screen.Fonts(i)
Next i
Do
  Sleep
Until Me Is Nothing

Sub cmd1_Click ()
  Dim i%
  lb2.Clear         ' Clear all items from the list.
  For i = 0 To lb1.ListCount - 1
    If lb1.Selected(i) Then
      lb2.AddItem lb1.List(i)
    End If
  Next i
End Sub

Sub lb1_Click
  ' The missing ListBox property: SelCount:
  Dim SelCount% = SendMessage(lb1.hWnd,
    LB_GETSELCOUNT, 0, 0)
  If SelCount = 0 && cmd1.Enabled
    cmd1.Enabled = False
  Else If SelCount > 0 && cmd1.Enabled = False
    cmd1.Enabled = True
  EndIf
EndSub
```

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

ComboBox Ocx

Purpose

Creates an **Ocx ComboBox** control in the current active form, window, or dialog.

Syntax

Ocx ComboBox name [= text$] [, id] [, x, y, b, h] [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

The control is a rectangle containing a list of strings (such as filenames) from which the user can select.

The **ComboBox** Ocx control has the following properties, methods, and events.

Properties

Appearance | BackColor | ForeColor | BorderStyle | Columns | DisableNoScroll | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | Height, Width | HelpContextID | hWnd | Index | IntegralHeight |ItemData | Left, Top | List | ListCount | ListIndex | MouseCursor | MouseIcon | MousePointer | Name | NewIndex | Parent | Scrollbars | Selected | Sorted |

Style | TabStop | Tag | Text | ToolTiptext | TopIndex | Visible | WhatsThisHelpID

Methods

AddItem | Clear | Find | FindExact | FindNext | InsertItem | Move | Refresh | RemoveItem | SetFocus | SetFont | TextHeight | TextWidth | ZOrder

Events

Click | DblClick | GotFocus | LostFocus | KeyDown, KeyUp | KeyPress | MouseDown | MouseUp | MouseMove | Scroll

There is no event which covers every eventually of the selected item being changed as the **Click** event only occurs when an object in the dropdown list is selected by using the mouse or by the up and down arrow keys, which means that occurences of the selected item being changed by physically typing in the value are missed. In the absence of a dedicated Change event, you can embed a call to **Click** in the **KeyUp** event which will effectively make the **Click** take on this role.

Remarks

The ComboBox object lacks some of the functionality that can be found in VB6 and most other programming languages which use it as an object; this can be overcome by using the *SendMessage()* API as follows:

- To find the position of Selstart (the cursor position) and SelEnd (the end of blocked text, if any) in the edit box, use `~SendMessage(cmb.hWnd, CB_GETEDITSEL, selstart, selend)`.

- To set the position of Selstart and SelEnd, use: `~SendMessage(cmb.hWnd, CB_SETEDITSEL, 0, MakeLong(selstart, selend))`.

- The editbox can be resized vertically by using `~SendMessage(cmb.hWnd, CB_SETITEMHEIGHT, -1, newheight%)`.

- The number of characters that can be entered into the edit box can be limited by using `~SendMessage(cmb.hWnd, CB_LIMITTEXT, limit, 0)`.

- A 'cue banner' can be added to the edit box using the code below:

```
Ocx ComboBox cmb = , 10, 10, 200, 22
Local cb$ = "[Cue Banner]"
Const CBM_FIRST = &1700
Const CB_SETCUEBANNER = (CBM_FIRST + 3)
~SendMessage(cmb.hWnd, CB_SETCUEBANNER, 0,
  UNI$(cb$))
Do : Sleep : Until Me Is Nothing

Function UNI$(ansi$) // Acknowledgements to
  Peter Heinzig
  Local lUni As Variant = CVar(ansi) : Return
    Peek$({V:lUni + 8}, Len(lUni) * 2) + #0
EndFunction
```

- The dropdown list of the ComboBox can be opened programmatically using `~SendMessage(cmb.hWnd, CB_SHOWDROPDOWN, True, 0)`, closed using `~SendMessage(cmb.hWnd, CB_SHOWDROPDOWN, False, 0)`, and the state of the dropdown list can be obtained using `~SendMessage(cmb.hWnd, CB_GETDROPPEDSTATE, 0, 0)`.

- The number of items visible when the dropdown list is shown can be altered using **MessageProc** event of the parent window or form to catch the CBN_DROPDOWN event of the ComboBox object as illustrated in the example below. A variation of this code can also be used to catch all the other ComboBox events as well.

```
// ComboBoxes which are to have full lists must
  have 'Full' somewhere in their Tag property.
// To set a 'Minimum Visible' limit, the string
  'MinVisxxx' (where xxx is the number of
  entries to show) must be somewhere in the Tag
  property.
// --- NB The value of the minimum visible
  entries can only be greater than 8.
Type COMBOBOXINFO
  - Long cbSize
  rcItem As RECT
  rcButton As RECT
  - Long stateButton
  - Long hwndCombo
  - Long hwndItem
  - Long hwndList
EndType
Type RECT
  - Long Left, Top, Right, Bottom
EndType
Const CB_GETCOMBOBOXINFO = 0x0164
OpenW 1
Local n As Int32
Ocx ComboBox cmb = "", 10, 60, 100, 22 :
  cmb.Tag = "Full" : For n = 1 To 20 :
```

```
  cmb.AddItem "Item no " & Iif(n < 10, " ", "")
  & Trim(n) : Next n
Ocx ComboBox cmb2 = "", 10, 120, 100, 22 :
  cmb2.Tag = "MinVis012" : For n = 1 To 20 :
  cmb2.AddItem "Item no " & Iif(n < 10, " ", "")
  & Trim(n) : Next n
Ocx ComboBox cmb3 = "", 10, 180, 100, 22 : For
  n = 1 To 20 : cmb3.AddItem "Item no " & Iif(n
  < 10, " ", "") & Trim(n) : Next n
Do : Sleep : Until Win_1 Is Nothing

Sub Win_1_MessageProc(hWnd%, Mess%, wParam%,
  lParam%, retval%, ValidRet?)
  Try
    If Mess% = WM_COMMAND And HiWord(wParam%) =
      CBN_DROPDOWN
      // Check to see if control is a ComboBox
      Local cn$ = Space(100) :
        ~GetClassName(lParam%, V:cn$, 100)
      If ZTrim(Mid(cn$, 2)) = "ComboBox"
        // Check to see if ComboBox list to be
          shown in full
        Local cb As Control : Set cb =
          OCX(lParam%)
        If InStr(Lower(cb.tag), "full") +
          InStr(Lower(cb.tag), "minvis") <> 0
          // Retrieve ComboBox Structure
            Information
          Local cbi As COMBOBOXINFO : cbi.cbSize
            = SizeOf(COMBOBOXINFO) :
            ~SendMessage(lParam%,
            CB_GETCOMBOBOXINFO, 0, cbi)
          // Retrieve ListBox rectangle
            coordinates
          Local lbr As RECT :
            ~GetWindowRect(cbi.hwndList, lbr)
          // Retrieve Item Count and Height
            values
          Local Int32 ct, h : h =
            SendMessage(cbi.hwndList,
            LB_GETITEMHEIGHT, 0, 0)
          If InStr(Lower(cb.tag), "full") <> 0 :
            ct = SendMessage(cbi.hwndList,
            LB_GETCOUNT, 0, 0)
          Else : ct = InStr(Lower(cb.tag),
            "minvis") : cn$ = Mid(cb.tag, ct, 9) :
            ct = Right(cn$, 3)
          EndIf
          If ct > 8 // If Item Count greater than
            default 8 entries
            // Calculate new height for ListBox
            h = (h * ct) + (Screen.cyBorder * 2)
            // Redraw ListBox
            ~MoveWindow(cbi.hwndList, lbr.Left,
              lbr.Top, lbr.Right - lbr.Left, h, 1)
            // Stop GB32 processing this message
            ValidRet? = True
            // Clear structures
            Clr cbi, lbr
          EndIf
        EndIf
      EndIf
    EndIf
  Catch
    // Include error message here if required
  EndCatch
EndSub
```

The CB_SETMINVISIBLE (&1701) and CB_GETMINVISIBLE (&1702) messages are available but GB32 overrides their actions when redrawing the dropdown list.

See Also

Ocx, OcxOcx

Animation, CheckBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 08/03/2018 by James Gaite}

Frame Ocx

Purpose

Creates an **Ocx Frame** control in the current active form, window, or dialog.

Syntax

Ocx Frame name [= text$] [, id], x, y, b, h [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

A **Frame** control provides an identifiable grouping for controls. You can also use a **Frame** to subdivide a form functionally-for example, to separate groups of **Option** controls.

To group controls, first draw the **Frame** control, and then draw the controls inside the **Frame**. In case of **Option** buttons make sure they belong to the frame; right click on the **Option** control and check 'Ocx on Frame'. By default, the controls are owned by the form and all **Option** controls on the form belong to the same group, event the controls outside the **Frame** control.

Properties

BackColor | BorderStyle | Caption | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | ForeColor | Height | HelpContextID | hWnd | Index | Left | MouseCursor | MouseIcon | MousePointer | Name | Parent | TabStop | TabStripIndex | Tag | Text | Top | ToolTiptext | Transparent | Visible | WhatsThisHelpID | Width

Methods

Move | Refresh | SetFont | TextHeight | TextWidth | ZOrder

Events

Click | DblClick | MouseDown | MouseUp | MouseMove

The mouse events occur when no other Ocx is positioned under mouse cursor. When **Transparent** = True a mouse event is only executed when the mouse is on a character pixel.

Example

```
Form frm = "Frame", , , 300, 300
'.BackColor = colBtnFace
Ocx Frame fr = "abc", 10, 10, 200, 200
.Transparent = False  ' default  is True
.BackColor = RGB(128, 0, 0)
Ocx Option opt(0) = "Option 1", 20, 30, 140, 24
Ocx Option opt(1) = "Option 2", 20, 60, 140, 24
Ocx Option opt(2) = "Option 3", 20, 90, 140, 24
Do
  Sleep
Loop Until Me Is Nothing
```

Remarks

A **Frame** is useful as a parent Ocx (**OcxOcx)** control. Other Ocx controls that can be used a parents are **Form**, **Image**, **TabStrip**, **Toolbar**.

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

Form Object

Purpose

A **Form** object is a window or dialog box that makes up part of an application's user interface.

Syntax

Form

Description

Forms are the foundation for creating the interface of an application. You can use forms to add windows and dialog boxes to your application. You can also use them as containers for items that are not a visible part of the application's interface. For example, you might have a form in your application that serves as a container for graphics that you plan to display in other forms.

A design a form using the Form Editor or create them in code. A form designed with the Form Editor is brought into the program using the **LoadForm** command. In code, forms are created by **Form**, **OpenW**, **ParentW**, **ChildW**, and **Dialog** (which see).

Forms have properties that determine aspects of their appearance, such as position, size, and color; and aspects of their behavior, such as whether or not they are resizable.

Forms can also respond to events initiated by a user or triggered by the system. For example, you could write code in a form's **Click** event procedure that would enable the user to change the color of a form by clicking it.

In addition to properties and events, you can use methods to manipulate forms using code. For example, you can use the **Move** method to change a form's location and size.

When designing forms, set the **BorderStyle** property to define a form's border, and set the **Caption** property to put text in the title bar. In code, you can use the **Hide** and **Show** methods to make forms invisible or visible at run-time.

Properties

Align | Appearance | AutoClose |AutoRedraw | BackColor | BkColor | BorderStyle | Caption | ControlBox | Controls | CurrentX | CurrentY | DrawMode | Enabled | Font | FontBold | FontItalic | FontName | FontSize | FontStrikethru | FontTransparent | FontUnderline | ForeColor | hDC~hDC2 | Height | HelpButton | HelpContextID | hMdiClientWnd | HScMax | HScMin | HScPage | HScPos | HScStep | HScTrack | hWnd | Icon | Image | Index | IsDialog | Left | MaxButton | MdiChild | MdiParent | MenuEnabled | MenuItem | MenuText | MinButton | MouseCursor | MouseIcon | MousePointer | Moveable | Name | OcxScale | OnTop | PaintLeft | PaintTop | PaintWidth | PaintHeight | Parent | Picture | PictureMode | PrintScroll | PrintWrap | ScaleHeight | ScaleLeft | ScaleMode | ScaleTop | ScaleWidth | ScrollBars | ShowInTaskBar | Sizeable | SmallIcon | StartUpMode | TabStripIndex | TabStop | Tag | ToolTipText | Top | Visible | VScMax | VScMin | VScPage | VScPos | VScStep | VScTrack | WhatsThisHelpID | Width | WindowState

The **PictureMode** property determinates how the Form picture is displayed (0 = default, 1 = Tile, 2 = stretched).

Methods

Activate | Adjust | Center | Close | Deactivate | Disable | DoClick | Enable | FullW | Hide | Invalidate | InvalidateAll | Maximize | MdiCascade | MdiGetActive | MdiActivate | MdiIconArrange | MdiNext | MdiPrev | MdiSetMenu | Minimize | Move | Owner | PixelsPerTwipX | PixelsPerTwipY | PrintForm | PrintFormHeight | PrintFormWidth | PrintPicture, PrintPicture2 | Refresh | Restore | Scale | ScaleX | ScaleY | SetFocus | SetFont | Show | SysMenuText | TextHeight | TextWidth | ToBack | ToTop | TwipsPerPixelX | TwipsPerPixelY | Validate | ValidateAll | WhatsThisMode | ZOrder

Events

Activate | Click | Close | DblClick | DDEWndProc | Deactivate | Destroy | DisplayChange | EndSession | GotFocus, LostFocus | HScroll | HScrolling | KeyDown, Keyup | KeyPress | Load | MciNotify | MenuEvent | MenuOver | Message | MessageProc | MonitorPower | MouseDblClick | MouseDown, MouseUp | MouseMove | MouseWheel | Moved | OnCtrlHelp | OnHelp | OnMenuHelp | Paint | QueryEndSession | Resize | ScreenSave | SysColorChange | SysMenuOver | VScroll | VScrolling | WinIniChange

Some properties are only valid for an **Ocx Form**, a form used as a control. For instance, a non-Ocx form cannot have a **Parent**, but an Ocx can and does.

Remarks

Note Setting the **BorderStyle** to 0 removes the border. If you want your form to have a border without the title bar or

Control-menu box, delete any text from the form's **Caption** property and set the form's **ControlBox** properties to False.

See Also

Form, Form(), LoadForm, OpenW, ChildW, ParentW, Dialog

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

MonthView Ocx

Purpose

Creates an **Ocx MonthView** control 'in' the current active form, window, or dialog.

Syntax

Ocx MonthView name [= text$] [, id%], [x, y, w, h] [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

The **MonthView** control enables you to create applications that let users view and set date information via a calendar-like interface.

The **Value** property of the **MonthView** control returns the currently selected date.

You can allow end users to select a contiguous range of dates by setting the **MultiSelect** property to **True**, and specifying the number of selectable days with the **MaxSelProperty**. The **SelStart** and **SelEnd** properties return the start and end dates of a selection.

You can customize a **MonthView** control's appearance in many ways. Various color attributes such as

MonthBackColor, **TitleBackColor**, **TitleForeColor,** and **TrailingForeColor** enable you to create a unique color scheme for the control.

You can display more than one month at a time (up to 12) in a **MonthView** control by setting the **MonthRows** and **MonthColumns** properties. The total of the **MonthRows** and **MonthColumns** properties must be less than or equal to 12.

The **MonthView** Ocx control has the following properties, methods, and events.

Properties

Appearance | BackColor | BorderStyle | Day | DayBold | DayOfWeek | DayVisible | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | ForeColor | Height | HelpContextID | HideSelection | hWnd | Index | Left | MaxDate | MaxSelCount | MinDate | Month | MonthBackColor | MonthColumns | MonthRows | MouseCursor | MouseIcon | MousePointer | MultiSelect | Name | Parent | ScrollRate | SelEnd | SelStart | ShowToday | ShowWeekNumbers | StartOfWeek | TabStop | Tag | TitleBackColor | TitleForeColor | Today | ToolTiptext | Top | TrailingForeColor | Value | Visible | VisibleDays | Week | WhatsThisHelpID | Width | Year

Methods

AboutBox | ComputeControlSize | HitTest | Move | Refresh | SetFocus | SetFont | ZOrder

Events

Click | DateClick | DateDblClick | DayClick | DblClick | GetDayBold | GotFocus | KeyDown, KeyUp | KeyPress | LostFocus | MouseDown | MouseUp | MouseMove | SelChange

Example

```
Mode StrSpace 0
OpenW 1, , , 570, 450
Ocx MonthView mvw = "", 10, 90, 0, 0 /* Width and
  Height are ignored
With mvw
  .MonthColumns = 2 : .MonthRows = 2
  .Value = Date
  .ForeColor = RGB(0, 0, 255)
  .MonthBackColor = colBtnFace
  .StartOfWeek = 1       ' Sunday
  .ShowToday = 1
  .ShowWeekNumbers = True
EndWith
Ocx Command cmd1 = "Restrict", 200, 10, 150, 25
Ocx Command cmd2 = "Multi Select", 200, 40, 150,
  25
Ocx Label lbl = mvw.Value , 200, 70, 80, 25
Ocx Label lblWeek = mvw.Week, 280, 70, 20, 25
Ocx Label lblWeekDay = mvw.DayOfWeek , 310, 70,
  20, 25
Debug.Show
mvw.SetFocus
Trace mvw.DayVisible(mvw.Value)
Trace mvw.ScrollRate
Do
  Sleep
Until Me Is Nothing

Sub cmd1_Click
  mvw.MinDate = #01.07.1998#
  mvw.MaxDate = #01.07.1999#
  mvw.Value = #17.04.1999#

Sub cmd2_Click
  mvw.MultiSelect = True
  mvw.MaxSelCount = 10
  mvw.Value = #01.01.1999#

Sub mvw_Click
  Debug.Print "Event _Click"

Sub mvw_DateClick(DateClicked As Date)
  Trace DateClicked

Sub mvw_DateDblClick(DateDblClicked As Date)
  Trace DateDblClicked

Sub mvw_DayClick(DayOfWeek%)
  Trace DayOfWeek

Sub mvw_DblClick
  Debug.Print "Event _DblClick"

Sub mvw_GetDayBold(StartDate As Date, Count%,
  State?())
  Debug.Print "Event _ GetDayBold"

Sub mvw_GotFocus
  Debug.Print "Event _GotFocus"

Sub mvw_KeyDown(Code&, Shift&)
  Debug.Print "Event _KeyDown (Param:
    ",Code&,":",Shift&,")"

Sub mvw_LostFocus
  Debug.Print "Event _LostFocus"
```

```
Sub mvw_SelChange(StartDate As Date, EndDate As
 Date)
 Trace StartDate
 Trace EndDate
 lbl = mvw.Value
 lblWeekDay = mvw.DayOfWeek
 lblWeek = mvw.Week
```

Known Issues

As at the time of writing (Win8/10), **DateDblClick** does not work; all that happens is that the **DateClick** event is called twice. See the DateClick page for a workaround.

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

TabStrip Ocx

Purpose

Creates an **Ocx TabStrip** control in the current active form, window, or dialog.

Syntax

Ocx TabStrip name [= text$] [, id%] [, x, y, w, h] [, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

A **TabStrip** control is like the dividers in a notebook or the labels on a group of file folders. By using a **TabStrip** control, you can define multiple pages for the same area of a window or dialog box in your application.

The control consists of one or more **Tab** objects in a **Tabs** collection. At run time, you can affect the **Tab** object's appearance by setting properties. You can also add and remove **Tab** objects at run time using methods.

The **Style** property determines whether the **TabStrip** control looks like push buttons (Buttons or Flat-Buttons) or notebook tabs (Tabs). At design time when you put a **TabStrip** control on a form, it has two notebook tabs. If the **Style** property is set to **tabTabs**, then there will be a border around the **TabStrip** control's internal area. When the **Style** property is set to **tabButtons**, no border is displayed around the internal area of the control, however, that area still exists.

To set the overall size of the **TabStrip** control, use its drag handles and/or set the **Top**, **Left**, **Height**, and **Width** properties. Based on the control's overall size at run time, Visual Basic automatically determines the size and position of the internal area and returns the Client-coordinate properties - **ClientLeft**, **ClientTop**, **ClientHeight**, and **ClientWidth**. The **MultiRow** property determines whether the control can have more than one row of tabs, the **TabWidthStyle** property determines the appearance of each row, and, if **TabWidthStyle** is set to **tabFixed**, you can use the **TabFixedHeight** and **TabFixedWidth** properties to set the same height and width for all tabs in the **TabStrip** control.

To contain the actual pages and their objects, you must use **Frame, Form,** or **Image** controls that match the size of the internal area which is shared by all **Tab** objects in the control. When **Frame** is used as a container (**OcxOcx** *tbs* **Frame** *frm*), it has the additional feature that BorderStyle = 0 and Transparent = 0. The coordinates specified in the **OcxOcx** command are ignored; the container is automatically sized to the **TabStrip** client coordinates.

The **Text**/**Caption** property of the **Frame** and **Form** is used as the title for the Tab. The **Image** control doesn't have a Caption property, and is less useful.

The **TabStrip** Ocx control has the following properties, methods, and events.

Properties

Appearance | BackColor | ClientHeight | ClientLeft | ClientTop | ClientWidth | Enabled | Font | FontBold | FontItalic | FontStrikethru | FontUnderline | FontName | FontSize | HotTracking | Height | HelpContextID | hWnd | ImageList | Index | Left | MouseCursor | MouseIcon | MousePointer | MultiRow | Name | Parent | Placement | ScrollOpposite | SelectedIndex | SelectedItem | Separators | Style | Tab | TabCount | TabFixedHeight | TabFixedWidth | TabMinWidth | Tabs | TabStop | TabWidthStyle | Tag | Top | Visible | WhatsThisHelpID | Width

The **TabCount** property returns the number of tabs. Short for **.Tabs.Count**.

Methods

Add | AddItem | Clear | HitTest | Item | Move | NextTab | PrevTab | Refresh | Remove | SetFocus | SetFont | TextHeight | TextWidth | ZOrder

Events

BeforeChange | Change | Click | KeyDown, KeyUp | KeyPress | MouseDown | MouseUp | MouseMove

Example

```
Const USE_ADD = 1
Form Hidden Center frm1 = "TabStrip", , , 400, 300
Ocx TabStrip tbs = , 20, 20, ScaleWidth - 40,
  ScaleHeight - 40
tbs.HotTracking = True
tbs.Placement = 1
If USE_ADD
  Ocx Frame fr1 = "Tab #1"
  Ocx Frame fr2 = "Tab #2"
  Ocx Frame fr3 = "Tab #3"
  Ocx Frame fr4 = "Tab #4"
Else
  ' See Remarks
  OcxOcx tbs Frame fr1 = "Tab #1"
  OcxOcx tbs Frame fr2 = "Tab #2"
  OcxOcx tbs Frame fr3 = "Tab #3"
  OcxOcx tbs Frame fr4 = "Tab #4"
EndIf
OcxOcx fr1 Option opt1 = "Option #1", 20, 20, 80,
  24
OcxOcx fr1 Option opt2 = "Option #2", 20, 50, 80,
  24
OcxOcx fr2 CheckBox chk1 = "Check #1", 20, 20, 80,
  24
OcxOcx fr2 CheckBox chk2 = "Check #2", 20, 50, 80,
  24
OcxOcx fr3 TextBox txt1 = "TextBox #1", 20, 20,
  280, 40
OcxOcx fr3 TextBox txt2 = "TextBox #2", 20, 130,
  280, 40
OcxOcx fr4 Command cmd1 = "NextTab", 90, 20, 80,
  24
OcxOcx fr4 Command cmd2 = "PrevTab", 90, 50, 80,
  24
If USE_ADD
  Dim tab As Tab
  tbs.Tabs.Add 1, , fr1.Caption , , fr1
  tbs.AddItem 2, , fr2.Caption, , fr2
  tbs.Add 3, , fr3.Caption, , fr3
  Set tab = tbs.AddItem(4, , , , fr4)
  tab.Caption = fr4.Caption
EndIf
' Creates ragged rows of tabs.
tbs.MultiRow = True
tbs.TabWidthStyle = tabNonJustified
frm1.Show
```

```
tbs(2).Selected = True
Do
  Sleep
Until Me Is Nothing

Sub tbs_Change
  Switch tbs.SelectedIndex
  Case 1 : opt1.SetFocus
  Case 2 : chk1.SetFocus
  Case 3 : txt1.SetFocus
  Case 4 : cmd1.SetFocus
  EndSwitch
End Sub

Sub tbs_BeforeChange(Cancel?)
  If MsgBox("Tab change allowed?", MB_OKCANCEL) =
    IDCANCEL
    Cancel? = True
  EndIf

Sub cmd1_Click
  tbs.NextTab

Sub cmd2_Click
  tbs.PrevTab
```

Remarks

When using the **OcxOcx** command to associate a container with a **TabStrip** control, the **Add[Item]** method is invoked implicitly. Also, the caption of the container is used for the **Tab** object **Text**.

A third way of creating **TabStrip** containers is by using the Form Editor. First create a Form with a **TabStrip** and then create a set of Forms that define the contents of each tab. Then in code:

```
LoadForm frmTabStrip Hidden
Do
  Sleep
Until Me Is Nothing

Sub frmTabStrip_Load
  LoadForm frm2 Hidden
  LoadForm frm3 Hidden
  LoadForm frm4 Hidden
  LoadForm frm5 Hidden
  tbs1.AddItem , , "GFA"  , , frm2
  tbs1.AddItem , , "Software", , frm3
  tbs1.AddItem , , "Technologies", , frm4
  tbs1.AddItem , , "GmbH", , frm5
  frmTabStrip.Show
EndSub

Sub tbs1_Change
  Switch tbs1.SelectedIndex
  Case 1 : chk2.SetFocus
  Case 2 : cmd2.SetFocus
  Case 3 : ed1.SetFocus
  Case 4 : cmd5.SetFocus
  EndSwitch
End Sub
```

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

TrayIcon Ocx

Purpose

Creates an **Ocx TrayIcon** control 'in' the current active form, window, or dialog.

Syntax

Ocx TrayIcon name [= text$] [, id%], [x, y, w, h] [, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

TrayIcon creates a taskbar notification icon. It places an icon of your choice into the System Tray that most often will display a ToolTip of your choice when the mouse is rested over it, will restore the application when clicked, and will display a popup menu when right-clicked.

Properties

Icon | Index | Name | Parent | Tag | ToolTipText | Visible

Events

MBDown | MBUp | MBDblClick | MMove

Syntax Events

Sub *TrayIcon*_**MBDown**(Button%)

Sub *TrayIcon*_**MBUp**(Button%)

These events occur when the user presses (**MBDown**) or releases (**MBUp**) a mouse button. The *Button%* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively.

Sub *TrayIcon*_**MBDblClick**(Button%)

Occurs when the user presses and releases a mouse button, then presses and releases it again over an object. The *Button%* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively.

Sub *TrayIcon*_**MMove**

Occurs when the user moves the mouse over the tray icon.

Example

```
OpenW 1
Ocx TrayIcon tic1
tic1.Icon = CreatePicture(LoadIcon(Null,
  IDI_APPLICATION))
tic1.ToolTipText = "Demo Application"
tic1.Visible = True
Do
  Sleep
Until Me Is Nothing
'

Sub Win_1_Moved
```

```
  If Win_1.WindowState = basMinimized _
    Win_1.Hide

Sub tic1_MBDown(Button%)
  Debug.Trace Button%
  If Button% = 2
    Local ret%
    DoEvents
    ret% = PopUp("&Open|-|E&xit", 0, 0, -3)
    Switch ret%
    Case 0 ' Restore / Open
      If Win_1.WindowState = basMinimized || _
        Win_1.Visible = False Then Win_1.Restore
    Case 2 ' Exit
      PostMessage Win_1.hWnd, WM_CLOSE, 0, 0
    EndSwitch
  EndIf
End Sub

Sub tic1_MBDblClick(Button%)
  Debug.Trace Button%
  If Button% = 1 Then _
    PostMessage Me.hWnd, WM_CLOSE, 0, 0
End Sub

Sub tic1_MBUp(Button%)
  Debug.Trace Button%
End Sub
```

Remarks

To make **PopUp** (*TrackPopupMenu* API) work properly in the context of a tray, you must first call *SetForegroundWindow* on the window that owns the popup. Otherwise, the menu will not disappear when the user presses Escape or clicks the mouse outside the menu. To find out more, search for Q135788 in MSDN. "This behavior is by design."

See Also

Ocx, OcxOcx

Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

Animation Ocx

Purpose

Creates an **Ocx Animation** control in the current active form, window, or dialog.

Syntax

Ocx Animation name = [text$] [, id] [, x, y, b, h] [, style%]

text$:control text
id%:control identifier
x, y, b, h:iexp
style%:the control styles

Description

The **Animation** control allows you to create buttons which display animations, such as .avi files, when clicked. The control can play only AVI files that have no sound. In addition, the Animation control can display only uncompressed .avi files or .avi files that have been compressed using Run-Length Encoding (RLE).

An example of this control is the file copy progress bar in Windows 95, which uses an **Animation** control. Pieces of paper "fly" from one folder to another while the copy operation executes. See example.

Properties

AutoPlay | BackColor | Enabled | Center | Height | HelpContextID | hWnd | Index | Left | Name | Parent | Tag | Top | ToolTiptext | Transparent | Visible | WhatsThisHelpID | Width

Methods

Close | Move | Open | Play | Seek | Stop | Refresh | ZOrder

Events

Click | DblClick | MouseDown | MouseUp | MouseMove | Start | Stop

Example

AnimOcx.g32 sample program.

Remarks

If you attempt to load an .avi file that includes sound data or that is in a format not supported by the control, an error is returned.

See Also

Ocx, OcxOcx

CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

UpDown Ocx

Purpose

Creates an **Ocx UpDown** control in the current active form, window, or dialog.

Syntax

Ocx UpDown name [= text$] [, id%] [, x, y, width, height] [, style%]

text$:control text
id%:control identifier
x, y, width, height:iexp
style%:the control styles

Description

An **UpDown** control has a pair of arrow buttons which the user can click to increment or decrement a value, such as a scroll position or a value in an associated control, known as a buddy control.

To the user, an **UpDown** control and its buddy control often look like a single control. The buddy control can be any control that can be linked to the **UpDown** control through the **BuddyControl** property, and usually displays data, such as a **TextBox** control or a **Command** control.

The text of the buddy control is determined by the UpDown OCX using the **Format** property.

The **UpDown** control can be positioned to the right (default) or left of its buddy control with the **LeftAlign** property. The **BuddyControl** property sets or returns the Ocx control used as the buddy control. The arrows may be positioned vertically (default) or horizontally with the **Horizontal** property.

The **Increment**, **Min**, **Max**, and **Wrap** properties specify how the **UpDown** control's **Value** property changes when the user clicks the buttons on the control. For example, if you have values that are multiples of 10, and range from 20 to 80, you can set the **Increment**, **Min**, and **Max** properties to 10, 20, and 80, respectively. The **Wrap** property allows the **Value** property to increment past the **Max** property and start again at the **Min** property, or vice versa.

The **Value** property specifies the current value within the range of the **Min** and **Max** properties. This property is incremented or decremented when the arrow buttons are clicked. The settings of the Min and Max properties determine whether the value is incremented or decremented when the arrow buttons are clicked.

The **ArrowKeys** property determines the purpose of the up and down arrow keys in the buddy control. When **ArrowKeys** is True, it causes the up-down control to increment and decrement the position when the UP ARROW and DOWN ARROW keys are pressed.

The **Change** event occurs whenever the **Value** property changes. The **Value** property can change through code, or by clicking the arrow buttons.

Properties

ArrowKeys | BuddyControl, | Enabled | Format | Height | Width | HelpContextID | Horizontal | hWnd | Increment | Index | Left | Top | LeftAlign | Max | Min | MouseCursor | MouseIcon | MousePointer | Name | Parent | Tag | ToolTiptext | Value | Visible | WhatsThisHelpID | Wrap

Methods

Move | Refresh | ZOrder

Events

Change | DownClick | MouseDown |MouseUp | MouseMove |UpClick

Example

```
Form frm1 = "UpDown", , , 200, 200
Ocx TextBox tbu = "??", 5, 5, 150, 24
.Appearance = 1
Ocx UpDown updn
.BuddyControl = tbu
.Max = 10
.Increment = 0.5
.Format = "0.0"
.Value = 2.5
Do
  Sleep
Until Me Is Nothing
```

See Also UpDownOcx.g32

Remarks

An **UpDown** control without a buddy control functions as a sort of simplified scroll bar.

See Also

Ocx, OcxOcx

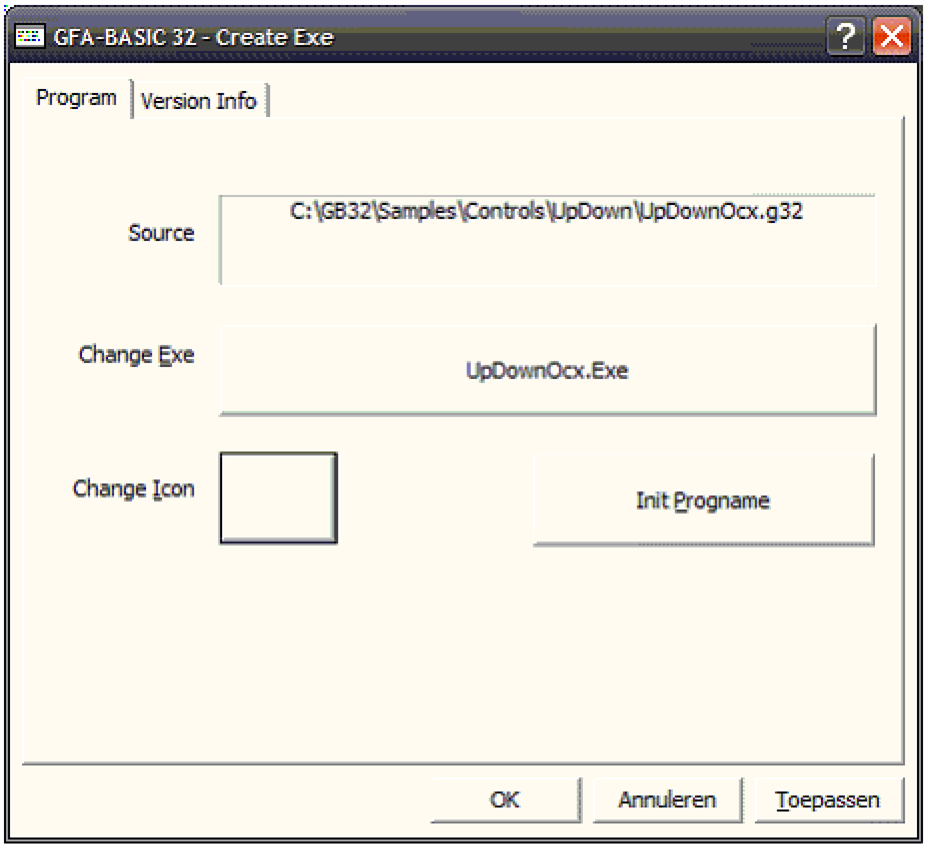
Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, TrayIcon, TreeView

{Created by Sjouke Hamstra; Last updated: 07/10/2017 by James Gaite}

Compile To Exe

After selecting Compile To Exe the following dialog box is displayed. Depending on the type of the source code (normal program, a GLL, or a library) two or three tabs are showed.

After filling in the dialog box click OK to start compiling. In addition, the information provided in the dialog box is saved in the project file in memory. Now the project is extended with the compile information, so it must be saved again to make the compile information persistent.


GFA-BASIC 32 - Create Exe

Program | Version Info

Source: C:\GB32\Samples\Controls\UpDown\UpDownOcx.g32

Change Exe: UpDownOcx.Exe

Change Icon

Init Progname

OK | Annuleren | Toepassen

Program tab

When compiling a normal program to a stand-alone executable this dialog box is displayed. This is a system property dialog box and shows the language of your Windows installation.

Source – Shows the name of the file currently loaded.

Change Exe – Shows the name of the stand-alone executable. To initialize the EXE name to the name of the source code file name, click the 'Init Progname' button. When you click the button displaying the EXE's name a file selection dialog box pops up which you can use the provide a different name.

Change Icon – Each EXE must contain an icon used to show with the file name in the Explorer.

Init Progname - Initializes the EXE name to the name of the source code file name.

Note – In case of compiling a GLL or LG32 library, you can still use the Program tab. The project can still be compiled to an EXE even if it is an editor extension or library. In case of an extension all **Gfa_** statements are ignored. It is possible to create a project that combines the functionality of a program and a GLL. For instance, a program might contain the logic to search for text in files. The project might then contain an interface to start the search from within a normal program. Additionally, the program may contain a GLL interface (keyboard shortcut or menu event) that starts the search logic as an editor extension.

Version info tab

In the 'Version Info' tab don't forget to press the small button with + to increment the file version number once a day.

To obtain help on the Version Info tab elements use the What's This Help button [?] in the title of the box.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Converting Data Types

GFA-BASIC 32 provides several conversion functions you can use to convert values into a specific data type. To convert a value to **Currency**, for example, you use the **CCur** function:

PayPerWeek = CCur(hours * hourlyPay)

Function	**Converts an expression to**
CBool	Boolean
CByte	Byte
CCur	Currency
CDate	Date
CDbl	Double
CShort	Short
CInt	Integer
CLong	Long
CLarge	Large
CSng	Single
CFloat	Single
CStr	String
CVar	Variant
CHandle	Handle

Note Values passed to a conversion function must be valid for the destination data type or an error occurs. For example, if you attempt to convert a **Large** to an **Integer**, the **Large** must be within the valid range for the **Integer** data type.

Floating point data conversion rounding towards zero.

CByteRZ, **CShortRZ**(), **CIntRZ**(), **CLongRZ**(), **CLargeRZ**().

See Also

CByte, etc...

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Set Command

Purpose

Assigns an object reference to a variable or property.

Syntax

Set *objectvar* **=** *objectexp* | **Nothing**

objectvar:name of variable or property
objectexp:any object expression

Description

When you use **Set** to assign an object reference to a variable, no copy of the object is created for that variable. Instead, a reference to the object is created. More than one object variable can refer to the same object. Because such variables are references to the object rather than copies of the object, any change in the object is reflected in all variables that refer to it.

objectexp is an expression consisting of the name of an object, another declared variable of the same object type, or a function or method that returns an object of the same object type.

The **Dim**, **Global**, **Public**, **Local**, and **Static** statements only declare a variable that refers to an object. No actual object is referred to until you use the **Set** statement to assign a specific object.

In GFA-BASIC 32 new instances of user interface **Ocx** objects are created with **Ocx**, **OcxOcx, LoadForm**, **Form**, and the window creation commands. OLE automation objects are created using **CreateObject** and **GetObject**. A mouse cursor object is created using **LoadCursor**, a **Picture** object with **CreatePicture** or **LoadPicture**.

```
Local pic As Picture
Set pic = LoadPicture("c:\pict.bmp")
```

Other intrinsic Ocx objects, like **DisAsm**, **Collection**, are created with the **New** keyword in the declaration. **Set** together with **New** cannot be used in GFA-BASIC 32, because GFA-BASIC 32 provides other means of creating object instances.

When **Nothing** is assigned to an object variable, the association of the object variable with the object is discontinued. Assigning **Nothing** to the object variable releases all the system and memory resources associated with the previously referenced object when no other variable refers to it.

```
Set pic = Nothing
```

Set Me is provided to assign a **Form** object to the **Me Form** object. **Set Me** redirects the output to the specified form without activating it.

```
Set Me = Win_1
```

Example

```
Dim dis As DisAsm
1
Set dis = CreateDisAsm()
2
```

```
dis.Addr = LabelAddr(1) // start address
Debug.Print dis.DisAsm          // disassembly of
  16 bytes
While dis.Addr < LabelAddr(2)
  Debug.Print dis
Wend
Debug.Show

Function CreateDisAsm() As DisAsm
  Dim dis As New DisAsm    // a new instance of
    disassembler
  dis.ByteFlag = True      // code bytes and Hex
    bytes
  dis.HexDump = False      // disassembly or a
    HexDump
  dis.HexDumpCount = 16    // bytes per line 1-32
    (16=default)
  dis.PreferHex            // addreses in hex format
  Set CreateDisAsm = dis
EndFunc
```

Remarks

See Also

Form, Me, OutPut

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

GetObject Function

Purpose

Returns a reference to an OLE object.

Syntax

Set *object* **= GetObject(**[*pathname*] [, *class*]**)**

object:Object exp

Description

The **GetObject** function accesses an OLE object from a file and assign the object to an object variable. Use the **Set** statement to assign the object returned by **GetObject** to the object variable. For example:

Global testobj As Object

Set testobj = GetObject("<path + file_name>", "program_name.object")

pathname specifies the full path and name of the file containing the object to retrieve. If *pathname* is omitted, *class* is required.

If *pathname* is a zero-length string (""), **GetObject** returns a new object instance of the specified type. If the *pathname* argument is omitted, **GetObject** returns a currently active object of the specified type. If no object of the specified type exists, an error occurs.

Some applications allow you to activate part of a file. Add an exclamation point (**!**) to the end of the file name and follow it with a string that identifies the part of the file you want to activate. For information on how to create this string, see the documentation for the application that created the object.

For example, in a drawing application you might have multiple layers to a drawing stored in a file. You could use the following code to activate a layer within a drawing called SCHEMA.CAD:

Set LayerObject = **GetObject**("C:\CAD\SCHEMA.CAD!Layer3")

If you don't specify the object's *class*, Automation determines the application to start and the object to activate, based on the file name you provide. Some files, however, may support more than one class of object. For example, a drawing might support three different types of objects: an **Application** object, a **Drawing** object, and a **Toolbar** object, all of which are part of the same file. To specify which object in a file you want to activate, use the optional ***class*** argument. For example:

```
Dim MyObject As Object
Set MyObject = GetObject("C:\DRAWINGS\SAMPLE.DRW",
  "FIGMENT.DRAWING")
```

In the example, FIGMENT is the name of a drawing application and DRAWING is one of the object types it supports.

Once an object is activated, you reference it in code using the object variable you defined. In the preceding example, you access properties and methods of the new object using the object variable *MyObject*. For example:

```
MyObject.Line 9, 90
MyObject.InsertText 9, 100, "Hello, world."
MyObject.SaveAs "C:\DRAWINGS\SAMPLE.DRW"
```

Note Use the **GetObject** function when there is a current instance of the object or if you want to create the object with a file already loaded. If there is no current instance, and you don't want the object started with a file loaded, use the **CreateObject** function.

If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times **CreateObject** is executed. With a single-instance object, **GetObject** always returns the same instance when called with the zero-length string ("") syntax, and it causes an error if the *pathname* argument is omitted. You can't use **GetObject** to obtain a reference to a class created with GFA-BASIC 32.

Example

Using **GetObject** with Excel objects

```
Dim x As Object
Set x = GetObject(, "Excel.Application")
' Excel must be running, otherwise an OLE
  Automation error occurs.
' The x refers to Excel.Application for the
  youngest instance of Excel.
Set x = GetObject("", "Excel.Application")
' Behaves like: Set x =
  CreateObject("Excel.Application").
Set x = GetObject("", "Excel.Sheet")
' Behaves like: Set x =
  CreateObject("Excel.Sheet")
Set x = GetObject("c:\Excel\test.xls")
Set x = GetObject("c:\Excel\test.xls",
  "Excel.Sheet")
' Each of these starts an invisible reference-
  independent
' instance of Excel, if it's not running,
' otherwise it uses the youngest existing
  instance.
' If the specified XLS file is not open, then it
' is opened as a hidden workbook, and is
' reference-dependent unless the command was
' executed as an Excel command. The x refers
' to the activesheet in the specified file.
Set s = GetObject("", "Excel.Chart")
' Behaves like: Set s =
  CreateObject("Excel.Chart")
Set s = GetObject("c:\Excel\test.xls",
  "Excel.Chart")
' Behaves like: GetObject("c:\Excel\test.xls")
' except that the workbook must contain at least
  one chart sheet.
```

Remarks

These are **ILLEGAL** :

```
Set s = GetObject(, "Excel.Sheet")
Set s = GetObject("c:\Excel\test.xls",
  "Excel.Application")
```

The built-in API function *GetObject* is renamed in **GetGdiObject** or **apiGetObject.**

See Also

Automation, CreateObject

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Lower/LCase and Upper/UCase Functions

Purpose

Converts a string to either upper or lower case

Syntax

$ = **Lower**[$](a$)

$ = **LCase**[$](a$)

$ = **Upper**[$](a$)

$ = **UCase**[$](a$)

a$:sexp

Description

Lower and **Upper** convert a string, including all accented letters, to lower and upper cases respectively.

LCase and **UCase** do the same but only for unaccented letters.

Non-letter characters are left unaffected by all of the above functions.

Example

```
Local a$ =
  "aàáâãäåæbcçdeèéêëfghiìíîïjklmnðñoòóôõöpqrstuvwxy
  z1234567890"
Debug.Show
Trace a$
Trace Upper(a$)
Trace UCase(a$)
Debug.Print
a$ = Upper(a$)
Trace a$
Trace Lower(a$)
Trace LCase(a$)
```

See Also

Xlate$()

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

ArrPtr Function

Purpose

Returns the address of a variable of any type.

Syntax

% = **ArrPtr**(x)

x:variable or field name

Description

ArrPtr returns the address of all variable types, except for arrays and string.

ArrPtr(a()) and **ArrPtr**(a$) return the addresses of array and string descriptors respectively. For a fixed string **ArrPtr** returns the first four bytes of the data. This function has no meaning for a fixed-string.

Example

```
OpenW # 1
Dim a(1), a$, n%
Print ArrPtr(a()) // prints the address of a()
  descriptor
Print *a$         // prints the address of a$
  descriptor
Print ArrPtr(n%)  // prints the address of n%
Print *n%         // prints the address of n%
```

Remarks

***** is synonymous with **ArrPtr**() and can be used instead.

See Also

Varptr(), V:

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Left Function

Purpose

Returns the first characters of a string expression.

Syntax

Left[$](a$ [,m [,fillchar]])

a$:sexp
m:integer expression
fillchar:iexp or sexp

Description

Left$(a$, m) returns the first m characters of the string expression a$. If m not given, the first character of a$ is returned. When m < 0 the string a$ is returned completely.

When m is greater than the number of characters in a$ (spaces and **Chr**$(0) are characters too!), the entire string returned. When **Left**() it takes a third parameter *fillchar*, it should specify the character to fill the return value when the source string does not hold enough characters. The *fillchar* might be ASCII value as well as a string containing the character to fill the string with.

Example

```
Print Left$("Hello GFA", 5)                // prints
 Hello
Print Left$("Hello GFA", 20)               // prints
 Hello GFA
```

```
Print Left$("Hello GFA", -1)              // prints
 Hello GFA
Print Left$("Hello GFA", 16, ".")         // prints
 Hello GFA.......
```

Remarks

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

See Also

String, Right(), Mid(), SubStr(), Mid

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

Right Function

Purpose

Returns the last m characters of a string expression.

Syntax

Right[$](a$ [,m [,fillchar]])

a$:sexp
m:integer expression
fillchar:iexp or sexp

Description

Right$(a$,m) returns the last m characters of the string expression a$. If m not given, the lasst character of a$ is returned. (m > 0)

When m is greater than the number of characters in a$ (spaces and **Chr**$(0) are characters too!), the entire string returned. When **Right** it takes a third parameter *fillchar*, it should specify the character to fill the return value when the source string does not hold enough characters. The *fillchar* might be ASCII value as well as a string containing the character to fill the string with.

Example

```
Print Right$("Hello GFA", 5)        // prints o GFA
Print Right$("Hello GFA", 20)       // prints Hello
 GFA
Print Right$("Hello GFA", -1)       // prints
Print Right$("Hello GFA", 16, ".") // prints
  .......Hello GFA
```

Remarks

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

See Also

String, Left$$, Mid$, SubStr

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Mid Command

Purpose

Moves a string expression of specified length to the chosen place in a character string.

Syntax

**Mid$**(a$, p% [,l%]) = b$

a$:svar
b$:sexp
p%, l%:integer expression

Description

**Mid$**(a$, p%, l%)=b$ moves l% characters from b$, to position p% (in a$) to a$. If l% is left out, again, as many characters as possible are moved from b$ to a$. The length and address of a$ are not changed.

Example

```
OpenW # 1 : Win_1.FontName = "courier new"
Local a$ = String$(15, "-")
Local b$ = "Hello GFA"
Print a$``Len(a$)     // Prints --------------- 15
Print b$``Len(b$)     // Prints Hello GFA 9
Mid$(a$, 3) = b$
Print a$``Len(a$)     // Prints --Hello GFA---- 15
Mid$(a$, 9, 5) = b$
Print a$``Len(a$)     // Prints --Hello Hello-- 15
```

See Also

Lset, Rset

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

Mid Function

Purpose

Starting from position p, returns the next m characters of a string expression.

Syntax

Mid[$](a$, p% [,m%])

a$:sexp
m%, p%:integer expression

Description

Mid$(a$, p%, m%) returns, starting from position p% (inclusive), up to m% characters of the string expression a$. If m% is not given, the whole string from position p% is returned.

Example

```
OpenW # 1
Print Mid$("Hello GFA", 7, 5)   // Prints GFA
Print Mid$("Hello GFA", 1, 5)   // Prints Hello
Print Mid$("Hello GFA", 3)      // Prints llo GFA
```

Remarks

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

See Also

String, Left$(), Right$(), SubStr()

{Created by Sjouke Hamstra; Last updated: 20/06/2017 by James Gaite}

SubStr Function

Purpose

Starting from position p, returns the next m characters of a string expression.

Syntax

SubStr[$](a$, p% [,m% [,char$]])

a$, char$:sexp
m%, p%:integer expression

Description

**SubStr$**(a$, p%, m%, char$) returns, starting from position *p%* (inclusive), up to *m%* characters of the string expression a$. If *m%* is not given, the whole string from position *p%* is returned. If m% specifies more characters than are present in *a$*, then the returned string is filled with character *char$.*

Example

```
Debug.Show
Local a$
a$ = "GFA Software Technologies GmbH"
Trace SubStr$(a$, 5, 9)          // Software
Trace SubStr(a$, 1, 3)           // GFA
Trace SubStr(a$, 5)              // Soft...
Trace SubStr(a$, 5, 35, "*")     // Software
  Technologies GmbH*********
```

Remarks

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

See Also

String, Left$(), Right$(), Mid()

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

InStr Function

Purpose

Searches a string expression for occurrence of a substring

Syntax

i = **InStr**(a, b [,m%] [, compmode])

i = **InStr**([m%], a, b [, compmode])

a, b: string expression or Variant expression
m%, compmode: iexp

Description

InStr() searches through the string or string in Variant expression a starting from position m% for the substring b. If m% is not given, the search starts from the first character in string a. The compare *compmode* indicates how the search for b inside a is to be performed. The *compmode* can take the same values as the **Mode Compare** statement.

compmode	***meaning***
0	binary compare (default)
1	case insensitive
-2	converts both strings to uppercase before searching
-3	converts both strings to lowercase before searching

If b isn't found or if b="" the command returns 0.

Example

```
Debug.Show
Trace InStr("Hello GFA", "ll", 2)          //
  Prints 3
Trace InStr("Hello GFA", "ll")             //
  Prints 3
Trace InStr("Hello GFA", "ll", 4)          //
  Prints 0
Trace InStr(0, "Hello GFABASIC", "LL", 1)  // 3
Trace InStr(0, "Hello GFABASIC", "LL", 0)  // 0
Trace InStr("Hello GFABASIC", "LL", 0, -2) // 3
Trace InStr("Hello GFABASIC", "LL", 0, -3) // 3
```

See Also

RInStr(), Mode

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

RInStr Function

Purpose

Searches a string expression for occurrence of a substring, optionally from a given position. If the substring is found, the position at which it begins is returned. If the substring is not found a 0 is returned.

Syntax

i = **RInStr**(a$, b$ [,m%] [, compmode])

i = **RInStr**([m%], a$, b$ [, compmode])

a$, b$: sexp
m%, compmode: iexp

Description

RInStr() searches through the string expression a$ starting from position m% for the substring b$. If m% is not given, the search starts from the first character in string a$. The compare *compmode* indicates how the search for b$ inside a$ is to be performed.

compmode	***meaning***
0	binary compare (default)
1	case insensitive
-2	converts both strings to uppercase before searching
-3	converts both strings to lowercase before searching

If b$="" the command returns 0.

Example

```
Debug.Show
Trace RInStr("Hello GFA", "ll", 2)          //
 prints 3
Trace RInStr("Hello GFA", "ll")             //
 prints 3
Trace RInStr("Hello GFA", "ll", 4)          //
 prints 0
Trace RInStr(0, "Hello GFABASIC", "LL", 1) // 3
Trace InStr(0, "Hello GFABASIC", "LL", 0)  // 0
Trace RInStr("Hello GFABASIC", "LL", 0, -2)// 3
Trace RInStr("Hello GFABASIC", "LL", 0, -3)// 3
```

See Also

Instr(), Mode

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Mirror Function

Purpose

Generates a value which is a mirror image of the given character expression.

Syntax

% = **Mirror**[**%**](x%)

% = **Mirror&**(x&)

% = **Mirror|**(x|)

% = **Mirror3**(x%)

l = **Mirror8**(xlarge)

$ = **Mirror**$(x$)

Description

Mirror[**%**] reverses the specified 32-bit integer value and returns a 32-bit value.

Mirror& reverses the specified 16-bit integer value and returns a 16-bit value.

Mirror| reverses the specified 8-bit integer value and returns a 8-bit value.

Mirror3 reverses the lower 24-bits of an integer value and returns a 32-bit value.

Mirror8 reverses the specified 64-bit integer value and returns a 64-bit value.

Mirror$ reverses the specified string value and returns it as a string

The arguments are converted to the expected type before the mirror operation is performed.

Example

```
Print Bin(Mirror
  (%11111110000000011111110000011), 32)
Print Bin(Mirror%
  (%11111110000000011111110000011), 32)
Print Bin(Mirror&
  (%11111110000000011111110000011), 32)
Print Bin(Mirror|
  (%11111110000000011111110000011), 32)
Print
 Bin(Mirror3(%11111110000000011111110000011),
 32)
Print
 Bin(Mirror8(%11111110000000011111110000011),
 64)
Print Mirror$("GFABasic32")
```

Prints

```
11000000111111100000000011111111
11000000111111100000000011111111
11111111111111111100000011111111
00000000000000000000000011000000
00000000110000001111111000000000
11000000111111100000000011111111111111111111111
11111111111111111
23cisaBAFG
```

Remarks

Use **_Swab** to swap bytes.

See Also

_Swab, _Swab8, _SwabL

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

VarType Function

Purpose

Returns a value indicating the subtype of a variable.

Syntax

% = **VarType**(varname)

varname:any variable name

Description

VarType returns one of the following values

basEmpty = 0	not initializes, an empty Variant
basNull = 1	Variant with the contents Null
basShort = 2	16 bit Integer (in VB vbInteger)
basLong = 3	Integer, Long, %, 32 bit
basInt = 3	Integer, Long, %, 32 bit
basSingle = 4	Floating point, single precision
basDouble = 5	Floating point, double precision
basCurrency = 6	Currency
basDate = 7	Date/Time
basVString = 8	String in Variant
basObject = 9	Object
basError = 10	Error value
basBoolean = 11	Boolean (True/False)
basVariant =	Variant (used only with arrays of

12	variants)
basByte = 17	Byte
basCard = 18	Integer, unsigned, 16 bit
basLarge = 20	Large, 64 Bit Integer
basType = 251	user-defiend type
basHash = 252	Hash
basFixedStr = 253	String with fixed length
basUnknown = 254	unknown
basString = 255	String ($), not as Variant
basArray = 8192	Array

The **VarType** function never returns the value for **basArray** by itself. It is always added to some other value to indicate an array of a particular type. The constant **basVariant** is only returned in conjunction with **basArray** to indicate that the argument to the **VarType** function is an array of type **Variant**. For example, the value returned for an array of integers is calculated as **basInt** + **basArray**, or 8194.

NOTE: This function does NOT work with native GFA Arrays and User-defined Types, despite the original documentation stating otherwise.

Example

```
Debug.Show
Local a As Card : Trace VarType(a)
Local o As Object : Trace VarType(o)
Local b As String : Trace VarType(b)
Local c As Double : Trace VarType(c)
Local i As Integer : Trace VarType(i)
Debug.Print
//
Local dd As Variant
dd = Array(1, 2, 4, "aaa", 17, Array(1, 2))
Trace VarType(dd(5)(1))
Trace VarType(dd)
Trace VarType(dd) - basArray
```

Remarks

This function is designed primarily to identify variable types in Variants and has been extended to do the same for simple native GFA variables. As neither GFA Arrays nor User-defined Types can be stored in a Variant, they can not be identified by this function.

See Also

TypeName, TypeOf, Gfa_Type

{Created by Sjouke Hamstra; Last updated: 07/07/2019 by James Gaite}

TypeName Function

Purpose

Returns a **String** that provides information about a variable.

Syntax

$ = **TypeName**[$](varname)

varname:variable

Description

TypeName returns a String naming the type of the variable, in contrast with VarType which only returns a constant representing the type.

Boolean	Boolean (True/False)
Byte	Byte
Card	Integer, unsigned, 16 bit
Currency	Currency
Date	Date/Time
Double	Floating point, double precision
Empty	not initialized, an empty Variant
Hash[]	Hash
Large	Large, 64 Bit Integer
Long	Integer, Long, %, 32 bit
Null	Variant or Handle with the contents Null
Single	Floating point, single precision
Short	16 bit Integer

String	(Fixed) String ($) including String in Variant
Variant	Variant (used only with arrays of variants)

Objects, Arrays and User-defined Types

When *varname* is an Object, the return value is the object type or, if no object has been assigned, then **Nothing**. Objects stored using OCX variables, Variants, the Picture and the Object variable types can be queried using this function.

With arrays, the function only works with **Arrays in Variants** and returns *type()* where *type* is the variable type of the array, e.g. a Double array would return "Double()".

This function does not work with User-defined Types nor elements of UDTs, despite advice otherwise in the original documentation.

Example

```
OpenW Hidden 1
Debug "General Variables"
Local ca As Card : Trace TypeName(ca)
Local i% : Trace TypeName(i%)
Local st$ : Trace TypeName(st$)
Local sh& : Trace TypeName(sh&)
Local e As Date : Trace TypeName(e)
//
Debug : Debug "Variants"
Local va As Variant : Trace TypeName(va)
va = 11122455.2255 : Trace TypeName(va)
va = "A string" : Trace TypeName(va)
va = Null : Trace TypeName(va)
//
```

```
Debug : Debug "Handles"
Local g As Handle : g = V:i% : Trace TypeName(g)
//
Debug : Debug "Objects"
Local f As Picture : Trace TypeName(f)
Set f = Win_1.PrintPicture : Trace TypeName(f) :
  Set f = Nothing
Local lbl As Label : Trace TypeName(lbl)
Ocx Label lbl1 : Set lbl = lbl1 : Trace
  TypeName(lbl)
Local obj As Object : Trace TypeName(obj)
Set obj =
  CreateObject("InternetExplorer.Application") :
  Trace TypeName(obj) : Set obj = Nothing
Local ova As Object : Trace TypeName(ova)
Ocx TextBox txt : Set ova = txt : Trace
  TypeName(ova)
//
Debug : Debug "Arrays in Variants"
Local aiv As Variant : aiv = Array(1, 2, 3) As
  Byte
Trace TypeName(aiv)
//
CloseW 1
Debug.Show
```

Remarks

This function is designed primarily to identify variable types in Variants and has been extended to do the same for simple native GFA variables. As neither GFA Arrays nor User-defined Types can be stored in a Variant, they can not be identified by this function.

See Also

TypeOf, VarType

{Created by Sjouke Hamstra; Last updated: 07/07/2019 by James Gaite}

Nothing Keyword

Purpose

The **Nothing** keyword is used to disassociate an object variable from an actual object.

Syntax

Set *Object* = **Nothing**

? *Object* **Is Nothing**
Boolean = **IsNothing**(*Object*)

Description

Use the **Set** statement to assign **Nothing** to an object variable.

Several object variables can refer to the same actual object. When **Nothing** is assigned to an object variable, that variable no longer refers to an actual object. When several object variables refer to the same object, memory and system resources associated with the object to which the variables refer are released only after all of them have been set to **Nothing**, either explicitly using **Set**, or implicitly after the last object variable set to **Nothing** goes out of scope.

All object variables are automatically cleared when they go out of scope. If you want the variable to retain its value across procedures, use a global variable, or create functions that return the object.

To check if an object has been set to **Nothing**, either the *Object* **Is Nothing** statement or **IsNothing** function can be used (they are interchangeable).

Example

```
OpenW 1
Dim dis As New DisAsm    // a new instance of
  disassembler object
// use the Disassembler
Set dis = Nothing        // release object
Print (dis Is Nothing)   // True
Print IsNothing(dis)     // True
Do
  Sleep
Until Win_1 Is Nothing   // [Or] Until
  IsNothing(Win_1)
```

Remarks

A reference to an OLE object can be stored in a an Object-type, which in fact is an IDispatch reference type, or a Variant.

See Also

Empty, Missing, Is, Null, Object, Set, Variant

{Created by Sjouke Hamstra; Last updated: 23/06/2017 by James Gaite}

IsDate Function

Purpose

Returns a **Boolean** value indicating whether an argument contains a Date type.

Syntax

Bool = **IsDate**(exp)

exp: Variant, Date, or String expression

Description

IsDate returns True if the expression is a date or is recognizable as a valid date; otherwise, it returns False. In Microsoft Windows, the range of valid dates is January 1, 100 A.D. through December 31, 9999 A.D.; the ranges vary among operating systems.

Example

```
Local z As Date = HmsToTime(110000, 20, 4000)
Print IsDate(z)              // True
Print IsDate(#16.12.1912#) // True
Local b$ = "31/12/2000"
Print IsDate(b$)          // -1 -> True
Local c As Variant = "23/25/1943"
Print IsDate(c)           // 0 -> False
```

See Also

IsDate, IsEmpty, IsError, IsMissing, IsNull, IsNumeric, IsObject

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

IsNumeric Function

Purpose

Returns a Boolean value indicating whether an expression can be evaluated as a number.

Syntax

Bool = **IsNumeric**(exp)

exp: any expression

Description

IsNumeric returns True if the expression is a **Variant** containing a numeric expression, a numeric expression, or a string expression.

IsNumeric returns False if expression is a date expression.

Example

```
Debug.Show
Local a As Variant, b#, c$
a = #22.12.1900#
Trace IsNumeric(a) //  0
b = 2222
Trace IsNumeric(b) // True
c = "10000"
Trace IsNumeric(c) // -1
c = "Hallo"
Trace IsNumeric(c) //  0
```

Remarks

IsNumeric tests whether the expression can be converted using OLE functions, which are language dependent (slower).

See Also

Val?, IsDate, IsEmpty, IsError, IsMissing, IsNull, IsNumeric, IsObject

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

IsObject Function

Purpose

Returns a Boolean value indicating whether a variable represents an Object.

Syntax

Bool = **IsObject**(varname)

Description

IsObject is useful in determining whether a Variant is of **VarType basObject**. This could occur if the **Variant** actually references (or once referenced) an object, or if it contains **Nothing.**

IsObject returns **True** if *identifier* is a variable declared with Object type, or if *identifier* is a OCX variable. **IsObject** returns **True** even if the variable has been set to **Nothing**.

Use error trapping to be sure that an object reference is valid.

Example

```
OpenW 1
Ocx TextBox tb2
Local a As Variant, b$
Print IsObject(tb2)// True
a = tb2
Print IsObject(a)  // -1
b$ = tb2
```

```
Print IsObject(b$) // False
```

See Also

IsDate, IsEmpty, IsError, IsMissing, IsNull, IsNumeric, IsObject

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

The Editor Extension Commands

There are approx. 130 special editor extension commands and functions to manipulate the IDE and source code. These commands begin with the prefix '**Gfa_**' and have in a normal program no effect. They are syntactically recognized; behave however like dummy commands. A project is marked as an editor extension when it contains at least one event Sub that starts with **Gfa_Ex_**, **Gfa_App**, or **Gfa_Run**, etc. The following sample shows a typical editor extension subroutine. The subroutine heading defines the keyboard shortcut invokes this routine: **Gfa_App_2**. The keyboard combination application key + 2 executes the Sub, because it starts with 'Gfa_'. The heading is not case sensitive.

Example: Switch quickly between two files

```
Sub Gfa_App_2    ' load MRU file #2
  If Gfa_Dirty Then Gfa_Save
  Gfa_LoadMRU 2
EndSub
```

The example is actually very useful, because it allows you to switch between the current project and the second project very quickly. First, the dirty status of the current project is checked and it is saved when the project has been changed. Then the second file from the most recently used (MRU) files list is loaded into the editor, making itself number one in the MRU list. The project that has been removed has now become number 2. Pressing App + 2 now will save the current project if dirty and reload the one just removed and has become #2.

If you've not created an extension before, you could copy the code above into a new project and then compile and install it.

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

GLL Keypress Event Subs

Syntax

Sub Gfa_Ex_?

Sub Gfa_App_?

Sub Gfa_App_S?

Sub Gfa_[S][C][A]F*

Description

Sub Gfa_Ex_?   (? as a placeholder for a alphanumeric key)

When a sub's name starts with **Gfa_Ex_** it is considered a Shift + Ctrl + key press event sub. The placeholder ? identifies an alphanumeric key (A-Z, 0-9). A GLL can contain up to 36 Gfa_Ex_? event subs.

Sub Gfa_App_?   (? as a placeholder for a alphanumeric key)

When a sub's name starts with **Gfa_App_** it is considered a App + key press event sub. The placeholder ? identifies an alphanumeric key (A-Z, 0-9). A GLL can contain up to 36 Gfa_App_? event subs.

Sub Gfa_App_S? (* as a placeholder for a alphanumeric key)

When a sub's name starts with **Gfa_App_S** it is considered a App + Shift + key press event sub. The placeholder ?

identifies an alphanumeric key (A-Z, 0-9). A GLL can contain up to 36 **Gfa_App_S**? event subs.

Sub Gfa_F*   (* = 2,8,9 for function keys F2, F8, and F9)

Sub Gfa_SF*   (* = 2,8,9,11,12 for function keys F2, F8, F9, F11, F12)

Sub Gfa_CF*   (* = 2,8,9,11,12 for function keys F2, F8, F9, F11, F12)

Sub Gfa_AF*   (* = 2,8,9,11,12 for function keys F2, F8, F9, F11, F12)

Sub Gfa_SCF*   (* = 2,8,9,11,12 for function keys F2, F8, F9, F11, F12)

Sub Gfa_SAF*   (* = 2,8,9,11,12 for function keys F2, F8, F9, F11, F12)

Sub Gfa_CAF*   (* = 2,8,9,11,12 for function keys F2, F8, F9, F11, F12)

Sub Gfa_SCAF*   (* = 2,8,9,11,12 for function keys F2, F8, F9, F11, F12)

Function key sub events. The following sub names identify the valid key combinations.

Gfa_F2, Gfa_F8, Gfa_F9, Gfa_SF2, Gfa_SF8, Gfa_SF9, Gfa_SF11, Gfa_SF12, Gfa_CF2, Gfa_CF8, Gfa_CF9, Gfa_CF11, Gfa_CF12, Gfa_SCF2, Gfa_SCF8, Gfa_SCF9, Gfa_SCF11, Gfa_SCF12, Gfa_AF2, Gfa_AF8, Gfa_AF9, Gfa_AF11, Gfa_AF12, Gfa_SAF2, Gfa_SAF8, Gfa_SAF9, Gfa_SAF11, Gfa_SAF12, Gfa_CAF2, Gfa_CAF8, Gfa_CAF9, Gfa_CAF11, Gfa_CAF12, Gfa_SCAF2, Gfa_SCAF8, Gfa_SCAF9, Gfa_SCAF11, and Gfa_SCAF12.

S = Shift, C = Ctrl, A = Alt, SCA = Shift + Ctrl + Alt

See Also

Gfa_Key, Gfa_KeyGet, Gfa_AddMenu

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

GLL Cursor Movement

Syntax

Gfa_Col [= c%]
Gfa_Line [= %]
Gfa_Left [n% = 1]
Gfa_Right [n% = 1]
Gfa_Down [n% = 1]
Gfa_Up [n% = 1]
Gfa_PageDown
Gfa_PageUp

Description

Gfa_Col [= c%] returns or sets the current column position.

Gfa_Col= moves the cursor to the column n (0 <= n <= line length) When n < 0, then 0 will be used, when n > line length, the line length will be used.

To set the cursor at the end of the line use **Gfa_Col = _maxInt**.

Gfa_Line [= %] returns or sets the current line. Moves the cursor to the specified line.

Gfa_Left [n% = 1] moves the cursor one or more characters to the left. The movement is not stopped at the beginning of the line. The parameter value can be negative, in which case the movement is in the opposite direction.

Gfa_Right [n% = 1] moves the cursor one or more characters to the right. The movement is not stopped at the end of the line. The parameter value can be negative, in which case the movement is in the opposite direction.

Gfa_Down [n% = 1] moves the cursor one or more lines down. The parameter value can be negative, in which case the movement is in the opposite direction.

Gfa_Up [n% = 1] moves the cursor one or more lines up. The parameter value can be negative, in which case the movement is in the opposite direction.

Gfa_PageDown moves the cursor one page down. When the cursor is in the bottom line the text is scrolled, otherwise the cursor is placed in the line at the bottom of the editor.

Gfa_PageUp moves the cursor one page up. When the cursor is in the top line (**Gfa_TopLine**) the text is scrolled, otherwise the cursor is placed in the top line.

Remarks

Note With **Gfa_Col=** the position is automatically clipped to the line length, in contrast with **Gfa_Right** and **Gfa_Left** that wrap the cursor to the next or previous line.

Gfa_Left, Gfa_Right, Gfa_Down, Gfa_Up is used to cancel a selection. The selection is canceled and the cursor is set at the beginning or the end of the selection, without moving the cursor out of the selection area. **Gfa_Left** and **Gfa_Up** set the cursor at the beginning. **Gfa_Right** and **Gfa_Down** ste the cursor at the end.

See Also

Gfa_Goto, Gfa_SelCol, Gfa_SelLine, Gfa_SelectAll, Gfa_IsSelection

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Gfa_SelCol, Gfa_SelLine, Gfa_SelectAll, Gfa_IsSelection

GLL Text Selection

Syntax

Gfa_SelCol [= c%]
Gfa_SelLine [= l%]
Gfa_SelectAll
? = Gfa_IsSelection

Description

The selection is the area between **Gfa_Line/Gfa_Col** and **Gfa_SelLine/Gfa_SelCol**. Every change made to **Gfa_Line** or **Gfa_Col** automatically resets **Gfa_SelLine** and **Gfa_SelCol** to the value in **Gfa_Line** and **Gfa_Col**. So, invoking **Gfa_Left** on a selection will remove the selection. When after changing **Gfa_Line** and/or **Gfa_Col** the range values **Gfa_SelLine** and/or **Gfa_SelCol** are newly set, the range between them is the new selection.

Gfa_SelCol [=] returns or sets the column at the start or the end of the selection.

```
col% = Gfa_SelCol

Gfa_SelCol = col%

Gfa_Col = _maxInt     ' select entire line
```

Gfa_SelCol = 0

Gfa_SelLine [=] returns or sets the specified line as the end of the selection.

line% = Gfa_SelLine

Gfa_SelLine = line%

Gfa_SelectAll Selects all text.

Gfa_SelectAll

Gfa_IsSelection returns **True** when a selection is available. **Gfa_IsSelection** is much faster then Len(**Gfa_Selection**). Internally, **Gfa_IsSelection** is the same as (**Gfa_SelLine** != **Gfa_Line** || **Gfa_SelCol** != **Gfa_Col**).

f? = Gfa_IsSelection

Gfa_Selection returns a string with the currently selected text, if any.

sel$ = Gfa_Selection

Example

```
// Get the selected text.
If Gfa_IsSelection Then sel$ = Gfa_Selection
```

Remarks

When **Gfa_SelCol** is greater than **Gfa_Col**, then the selection starts at **Gfa_Col** and ends at **Gfa_SelCol**. **Gfa_SelCol** can be smaller as **Gfa_Col**, so that the selection starts at **Gfa_SelCol** and ends at **Gfa_Col**.

See Also

Gfa_Cut, Gfa_Copy, Gfa_CopyRtf, Gfa_CopyPre

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_Cut, Gfa_Paste, Gfa_Copy, Gfa_CopyRtf, Gfa_CopyPre

GLL Clipboard commands

Syntax

Gfa_Cut

Gfa_Paste

Gfa_Copy

Gfa_CopyRtf

Gfa_CopyPre [head$, tail$, flag]

Description

Gfa_Cut copies the selected text to the clipboard and deletes it from its original location.

Gfa_Paste inserts the clipboard contents at the current location.

Gfa_Copy copies the selected text to the clipboard.

Gfa_CopyRtf copies the selection in RTF format preserving syntax colouring, font, and indention. The contents of the clipboard can be pasted in a RTF compatible word processor.

Gfa_CopyPre [head$, tail$, flag] copies the source code in CF_TEXT format with HTML-coding to the clipboard. When a selection is available, the selection is copied, otherwise the entire source code text.

The optional parameter Head$ may contain HTML code that is inserted before the <pre> tag.

The optional parameter Tail$ may contain HTML code that is appended to the closing <\pre> tag. (For instance, "</BODY></HTML>", when the source code is added at the end of a HTML page).

The optional integer flag% species whether to include the procedure separation line which is used in the editor to visually separate procedures. When flag = 1 adds </pre><hr><pre> (a HTML-dividing line). When flag = 0 there will be no dividing line.

```
Sub Gfa_App_P
  Gfa_CopyPre
EndSub
```

When this piece of code is selected and copied to the clipboard it is placed between <pre> tags and font tags are inserted. GFA-BASIC 32 puts the following string on the clipboard:

"<pre>Sub <Font Color=800000>Gfa_App_P" #13#10 _

" </Font>Gfa_CopyPre" + #13#10 _

"EndSub" #13#10 "</pre>"

The string starts with the <pre> tag followed by the Sub keyword. Since the procedure name is colored in red, the

<Font Color=800000> tag is inserted. The font coloring is disabled after the CRLF and the rest of the code is colored in the default color. The snippet is ended with the <\pre> tag.

Example

```
Sub Gfa_App_P
  Dim head$ = "<HTML><HEAD><META HTTP-EQUIV" _
    "=""Content-Type"" content=""text/html;" _
    " charset=iso-8859-1"">" #13#10 _
    "<TITLE>" & App.Name & "</TITLE></HEAD>
      <BODY>"#13#10 _
    "<H1 align=Center>//" & App.Name & "</h1><hr>"
  Dim tail$ =  "</BODY></HTML>"#13#10
  Gfa_CopyPre head$, tail$, 1
End Sub
```

Remarks

To save the entire source code use **Gfa_SaveFile**.

See Also

Gfa_SelCol, Gfa_SelLine, Gfa_SelectAll, Gfa_IsSelection, Gfa_SavePreFile

Gfa_Undo, Gfa_CommentBlock

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Gfa_Text

Syntax

s$ = **Gfa_Text**

Gfa_Text = s$

s$ = **Gfa_Text**(n)

Description

Gfa_Text Returns or sets the text of the current line, previous text will be deleted and leading spaces are ignored. **Gfa_Text** is more or less equivalent to **Gfa_Col = _maxInt** : **Gfa_SelCol** = 0 : **Gfa_Replace** s$.

Gfa_Text(n) returns the text of line n. The text is returned without leading spaces, even when the line is displayed indented. The indenting is performed dynamically when a line is written to the screen. The code text is reformatted continuously while editing. There is no **Gfa_Text**(n)= command.

See Also

Gfa_Insert, Gfa_Replace, Gfa_DeleteLines, Gfa_InsertLines

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Find & Replace

Syntax

Gfa_FindDlg
Gfa_FindNext
Gfa_FindPrev
Gfa_FindText [= $]
Gfa_ReplaceDlg
Gfa_ReplaceText [= $]
Gfa_ReplaceNext
Gfa_ReplaceAll

Description

Gfa_FindDlg displays the Find dialog box. The dialog always defaults the search text to the word the cursor is currently in.

Gfa_FindNext searches for a text string from the selected text's end to the end of the text document. If the text is found, the edit point is moved to the beginning of the match. Otherwise, the edit location is unchanged. This function is equivalent to <Find Next> in the IDE Edit menu. When the text to search for isn't set prior, either by use of the Find dialog box or by **Gfa_FindText=**, the Find dialog box is displayed. By default, the search is not case sensitive and does not search for whole words.

Gfa_FindPrev searches for a text string from the current position towards the start of the text document. There is no menu entry for **Gfa_FindPrev**, the only way to invoke this command by using Shift+F3.

Gfa_FindText [=] returns or sets the text to search. Sets the text to search used with **Gfa_FindNext** and **Gfa_FindPrev** (max. 256 characters). The search is not case sensitive and does not search for whole words.

Gfa_ReplaceDlg displays the Find and Replace dialog box. The dialog always defaults the search text to the word the cursor is currently in. The search direction is always towards the end of the text document. The text to search for can be obtained using **Gfa_FindText** and the replacement text with **Gfa_ReplaceText**.

Gfa_ReplaceText [=] returns or sets the replacement text used with **Gfa_ReplaceNext** (max. 256 characters).

Gfa_ReplaceNext searches and replaces a text string. **Gfa_ReplaceNext** searches for a text string (**Gfa_FindText**) from the current text position to the end of the text document. If the search text is found, the edit point is moved to the beginning of the match and the text is replaced with the replacement text (**Gfa_ReplaceText**). If no replacement text is specified the Find and replace dialog box is displayed.

Gfa_ReplaceNext is equivalent to pressing Ctrl+F3. The inverse operation; search and replace towards the start of the text document is possible with Shift+Ctrl+F3, but has no equivalent editor extension command. However, there is an easy workaround by sending the command ID value for the accelerator key.

```
' Search & Replace Previous
'
Const accCtrlShiftF3 = 0x432
SendMessage Gfa_hWnd, WM_COMMAND,
 MakeWParam(accCtrlShiftF3, 1), 0
```

Gfa_ReplaceAll searches and replaces all occurrences of a text string. Searches and replaces all occurrences of a text string. The text to search for can be set using the Find and Replace dialog box or by assigning it to **Gfa_FindText**. The replacement text is either provided in the dialog box or set with **Gfa_ReplaceText**.

Example

Remarks

See Also

{Created by Sjouke Hamstra; Last updated: 22/10/2017 by James Gaite}

Bookmarks

You can set named or unnamed bookmarks to mark frequently accessed lines in your source file. Once a bookmark is set, you can use menu or keyboard commands to move to it. You can remove a bookmark when you no longer need it.

From inside the editor a bookmark is toggled with Shift + Ctrl + Up or Shift + Ctrl + Down. To move to an unnamed bookmark use Ctrl + Up or Ctrl + Down to jump to the previous or next bookmark respectively.

To set numbered bookmarks in the IDE use Ctrl + K to display a popup menu or click in the left margin of the editor. Even so, to go to a numbered bookmark use Ctrl + Q to invoke the GoTo Mark popup menu or right click in the editor's margin.

Syntax

set? = **Gfa_BookMark**[(line%)] **Gfa_BookMark**(line%) = set?
Gfa_BookMark [n]

Gfa_NextBookMark [f% = 0]
line% = **Gfa_NextBookMark**([start%] [,f%=0])

line% = **Gfa_Mark**(i%) (i in 0..9)
Gfa_Mark(i%) = line%

Description

Gfa_BookMark function tests if a bookmark is set in the current line.

Gfa_BookMark(line) function tests if a bookmark is set in the specified line. When line=-1, the current line is assumed: **Gfa_BookMark** ó **Gfa_BookMark**(-1).

Gfa_BookMark(line)= sets or clears a bookmark in the specified line. When line% = -1, the current line is assumed.

```
// Toggle a bookmark

Sub Gfa_Ex_B //Shift+Ctrl+B
  Gfa_BookMark(-1) = Not Gfa_BookMark
EndSub
```

The **Gfa_BookMark** command jumps to the next or previous bookmark.

Gfa_BookMark [0] jump to next book mark
Gfa_BookMark -1 jump to previous book mark

Gfa_NextBookMark jumps to the next or previous unnamed bookmark. When the parameter f is specified and its value <> 0 the editor jumps to the previous bookmark. This command is circular, so that when the first bookmark is reached the search is restarted from the end of the source text.

Gfa_NextBookMark() function returns the line number with the next unnamed bookmark. When start is omitted or start <= 0 then the next bookmark is searched starting from the current position.
When the parameter f is specified and its value <> 0 the previous bookmark is searched.
This function is circular, so that when the first bookmark is

reached the search is restarted from the end of the source text.
The result is 0 when no bookmark is set at all, except for the current line.

Gfa_Mark(i) [=] returns or sets the line that contains the numbered bookmark. The return value of **Gfa_Mark**(i) is negative when the mark number *i* is out of range. A numbered mark is removed by assigning a negative value to the numbered mark.

Example

See Also

Gfa_CtrlK, Gfa_CtrlQ

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Gfa Ctrl + Key Shortcuts

Syntax

Gfa_CtrlK
Gfa_CtrlQ
Gfa_CtrlN
Gfa_CtrlP
Gfa_CtrlO
Gfa_CtrlY
Gfa_CtrlU

Description

Gfa_CtrlK emulates a Ctrl + K key press, which brings up the Set Bookmark popup menu.

Gfa_CtrlQ emulates a Control + Q key press, which brings up the Goto bookmark popup menu.

Gfa_CtrlN emulates a Control + N key press, which inserts a new line above the current line. This is the same as **Gfa_InsertLines** 1.

Gfa_CtrlP emulates a Control + P key press, which deletes the text from to current position to the end of the line.

Gfa_CtrlO emulates a Control + O key press, which inserts previously deleted text with Ctrl + P at the current position.

Gfa_CtrlY emulates a Control + Y key press, which deletes the current line.

Gfa_CtrlU emulates a Control + U key press, which inserts a previously deleted line with Ctrl + Y above the current line.

Example

See Also

Gfa_InsertLines, Gfa_BookMark

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

New, Loading, and Printing

Syntax

Gfa_New

Gfa_DoNew

Gfa_Print

Gfa_Load

Gfa_LoadFile filename$ [,fMru% = 0] (filename As String, fMru as Int)

Gfa_LoadMRU n% (n in 1..9)

Gfa_MergeFile filename$

Description

Gfa_New executes the File | New menu item creating a new project. The current project is removed and the new project gets the name 'noname.g32'. When the current project has the **Gfa_Dirty** status the project can be saved first.

Gfa_DoNew creates a new project without checking the **Gfa_Dirty** status. There is no correspondence menu item for this command.

Gfa_Print starts printing the selection or, when no text has been selected, the entire source code using the settings from the Properties dialog box.

Gfa_Load invokes the menu command <File | Load>. **Gfa_Load** displays a file-open dialog box, showing the current active directory.

Gfa_LoadFile loads the specified file without displaying a file-open dialog box and without providing a save option when the current project has changed.
When the optional parameter fMru% species a value other than zero (fMru <> 0) the filename is added to MRU list. When fMru = 0 or when the parameter is omitted the filename is not added to MRU list. In addition, the name of the file is not shown in the caption of the IDE and **Gfa_FileName** is "Noname.g32" or "OhneName.g32".

Gfa_LoadMRU loads the file with the specified MRU number. Loading the file updates the MRU list and puts the file at the top of the MRU list. The MRU list consists of the 9 menu entries in the File submenu that reflects the MRU file list in the register. In HKCU\Software\GFA\Basic the keys file1 to file9 specify the MRU list. The IDE updates the File submenu not before the File submenu is activated, in which case the IDE receives the WM_INITMENUPOPUP message.

Gfa_LoadMRU gets the filename from the <File> menu with the *GetMenuString* API function, not from the register. It turns out the File menu is not always up to date, for instance just after the start of the IDE when the File submenu isn't activated yet. The next example shows a modified MRU load.

Gfa_MergeFile inserts a GFA-BASIC 32 (g32) file or an ASCII file at the current position. When the file is a GFA-BASIC 32 project file, the form data and ‘:Files’ resources are inserted as well.
The IDE does not provide any means to invoke this

command. The example shows how to add the file insertion functionality.

Example

```
Sub LoadMRU(Optional FileNo As Int = 1)
  ' The Gfa_loadMRU command gets the filename from
    the menu entries
  ' in the File submenu. However, the submenu items
    are not updated
  ' before the File submenu is actually selected,
    in which case
  ' the MRU filelist is read from the registry.
    Occasionally, the
  ' file to load isn't added to the submenu yet.
    Instead we must get the
  ' MRU file name from register directly.
  Local String MRU = Gfa_Setting("File" $
    Dec(FileNo))
  If Exist(MRU)
    Gfa_LoadFile MRU, 1            ' 1=add to MRU,
      0=don't
  EndIf
EndSub
```

The following example adds a menu item to the Extra menu to create an event sub to insert a file using Gfa_MergeFile:

```
Sub Gfa_Init
  Global Int IdxMerge = Gfa_AddMenu("Insert file
    ...", menuMerge)
  Gfa_MenuDesc(IdxMerge) = "Inserts the contents of
    the specified file at the current location."
End Sub
```

```
Sub menuMerge(i%)
  Local fname As String
  FileSelect # "Insert file", CurDir + "\*.*", "",
    fname
  If Exist(fname)
    Gfa_MergeFile fname
  EndIf
EndSub
```

Remarks

See Also

Gfa_Save

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Save Project File

Syntax

Gfa_Save

Gfa_SaveAs

Gfa_SaveFile filename$ [,f% = 0]

Gfa_SaveRtf

Gfa_SaveRtfFile filename$

Gfa_SavePreFile filename$ [,head$, tail$, flag]

Description

Gfa_Save saves the current project. If it hasn't been saved before the Save dialog box is displayed to give the project a filename. Internally **Gfa_SaveAs** is invoked.

Gfa_SaveAs displays the Save As dialog box to save the project a (different) filename. The new filename is reflected in the title bar if the GFA-BASIC 32 IDE. The GFA-BASIC 32 most recent used files list in the register is updated with the new name.

Gfa_SaveFile saves the current project under the specified filename. The default behavior (f = 0) is not to update the MRU list, meaning the current project is not renamed. This allows for automating a backup saving at regular time intervals without disturbing the current settings. In particular, the current line is not parsed before the file is saved.

When f <> 0 the project is given the specified name and the MRU list is updated. Nothing happens when the filename argument is an empty string.

Gfa_SaveRtf displays the Saves As dialog box to save the source code text in RTF format. Displays the Save As dialog box to give the project a (different) filename to save the code text in RTF-format. When a selection is available, only the selection is saved, otherwise the entire code. This function is also available in the Edit menu.

Gfa_SaveRtfFile *filename$* saves source code text in RTF format in the specified file. Saves the source code RTF-formatted in the specified file. When a selection is available, the selection is saved, otherwise the entire code.

Gfa_SavePreFile *filename$* Saves the source code HTML-formatted in the specified file. When a selection is available, the selection is saved, otherwise the entire code. This saves the source code text in HTML code between <pre> … <\pre> tags.

- The optional parameter Head$ may contain HTML formatted text that is inserted before the <pre> tag.

- The optional parameter Tail$ may contain HTML code that is appended to the closing <\pre> tag. (For instance, "</BODY></HTML>", when the source code is added at the end of a HTML page).

- The optional integer flag% species whether to include the procedure separation line which is used in the editor to visually separate procedures. When flag = 1 adds </pre><hr><pre> (a HTML-dividing line). When flag = 0 there will be no dividing line.

See Gfa_CopyPre for an example.

Example

```
// Save before Test (syntax check)

Sub Gfa_F2
  Const ID_ShiftF5 = 0x429
  If Gfa_Dirty Then Gfa_Save
  PostMessage Gfa_hWnd, WM_COMMAND,
    MakeWParam(ID_ShiftF5, 1), 0
EndSub
```

Here the file is saved if it has been changed since the last saving. Then WM_COMMAND with the accelerator ID of Shift + F5 is posted to the message queue of the IDE.

```
Sub Gfa_Minute
  'autosave
  Const Delay1 = 10                ' ten minutes timer
  Static Int Timer1 = Delay1
  If Timer1 = 0
    If Gfa_Dirty
      Local t$ = TempDir & "temp.g32"
      Gfa_StatusText = "Autosaved to " + t$
      Gfa_SaveFile t$             ' no filename change
    EndIf
    Timer1 = Delay1
  EndIf
  If Timer1 > 0 Then Timer1--
EndSub
```

Remarks

Before executing **Gfa_Save**, **Gfa_SaveAs,** or **Gfa_SaveFile** "Name", 1, the current line, when changed, is parsed (by invoking **Gfa_Update)**. In case of a syntax error, the saving process is aborted. This is not the case when using **Gfa_SaveFile** "name", 0. When currently a line is being edited, the original line is saved, not the new edits.

See Also

Gfa_Dirty, Gfa_Load, Gfa_LoadMRU

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Gfa_Fold & Gfa_ProcLine

GFA-BASIC 32 is a procedure oriented language as C is. The IDE keeps record of the procedures and allows them to be collapsed. The information and status of the current procedure (the procedure containing Gfa_Line) can be obtained using the procedure functions.

Syntax

procname$ = **Gfa_Proc**

line% = **Gfa_ProcLine**

lines% = **Gfa_ProcLineCnt**

folded? = **Gfa_IsFold** [(n <= 0)]

Gfa_Fold [n = 1]

Description

Gfa_Proc returns the name of the current subroutine. When the current line is part of the main the program, **Gfa_Proc** returns an empty string.

Gfa_ProcLine returns the number of the first line of the current procedure.

Gfa_ProcLineCnt returns the number of the lines of the current procedure.

Gfa_IsFold tests whether the current line is located in a folded procedure. When n<=0 or is omitted the current line is tested, otherwise the specified line is tested.

Gfa_Fold folds or unfolds a procedure.

Gfa_Fold (or **Gfa_Fold** 1) folds the current procedure (collapse). **Gfa_Line** is reset to the first line of the procedure.

Gfa_Fold 0 unfolds the current procedure.

Gfa_Fold -1 toggles the current folding state of the procedure (same as F11).

Example

```
// Folding Procedures

Sub Gfa_Ex_F                ' Shift+Ctrl+F
  Gfa_Fold -1              ' Toggle folding
  If Gfa_IsFold
    Debug Gfa_Line & " in folded sub"
  EndIf
EndSub
```

Remarks

There is no editor extension function to fold all procedures.

See Also

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_Changed, Gfa_Update

Syntax

changed? = **Gfa_Changed**

Gfa_Changed = set?

Gfa_Update

Description

Gfa_Changed, **Gfa_Changed**= gets the current status of the current line. When True the line is currently being edited, otherwise nothing is changed in the current line.

Sets the change status of the current line.

When set to True the line will be parsed (**Gfa_Update**) when the cursor leaves the line. Updating can be prevented by setting **Gfa_Error** = True.

Setting **Gfa_Changed** to False will restore the old contents of the line.

Gfa_Update invokes the line parser responsible for error checking, syntax coloring and reformatting the code. When the line contains an error, depending on the setting in the properties dialog box for the editor, a message box is displayed. Otherwise, the line is marked as erroneous by invoking **Gfa_Error**. The line is then displayed in the error syntax color (red).

Example

Remarks

Syntax Checking Explained - As soon as you start editing a line, the line is copied to a temporary edit buffer, which replaces the original line on the screen. This process is visualized by switching the syntax coloring of the line to black (usually, but it can be changed in Properties dialog box). Internally a flag is set to indicate that the current line is being changed or edited. The function **Gfa_Changed** reflects the editing state of the current line.

When the cursor leaves a line that has been edited, the **Gfa_Update** statement is executed to parse the line and to look for errors. When there are syntax errors the line is marked as erroneous and is displayed in the error color red (usually, but this can be changed in Properties dialog box).

See Also

Gfa_Error, Gfa_NextError, Gfa_PrevError

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Gfa_Run, Gfa_IsRun, Gfa_Cleanup

Syntax

Gfa_Run

? = **Gfa_IsRun**

Gfa_Cleanup

Description

Gfa_Run should start the current loaded project (F5). Compiles the project first and executes it when no errors are found. GLL and Library projects cannot be executed. Due to a bug this command doesn't work

Gfa_IsRun the function **Gfa_IsRun** returns true when a program is running.

Gfa_Cleanup Cleanups still active resources after ending the program. Useful after a sudden break down of the program when there are still windows or files open. **Gfa_Cleanup** closes all handles and files that are created using GFA-BASIC 32 commands (not WINAPI).

Example

See Also

Gfa_OnRun, Gfa_OnEnd

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Gfa_DbStep, Gfa_DbOn/Gfa_DbOff

Syntax

Gfa_DbStep

Gfa_DbOn

Gfa_DbOff

Description

Gfa_DbStep advances the program to the next statement. **Gfa_DbStep** is necessary for a custom debugger to step through a program. This command should be used in the **Gfa_DebOn** 1 and 2 procedure.

When using the tray icon debugger **Gfa_DbStep** is automatically invoked when the tray icon is clicked with the left mouse button.

Gfa_DbOn shows the debug arrow in the edit window. **Gfa_DbOff** hides the debug arrow in the edit window.

See Also

Gfa_DebMenu,Gfa_DebMenu

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Gfa_Var Object

Syntax

Dim v As **Gfa_Var**

Description

A **Gfa_Var** item provides read-only properties that allow you to get information about the variable like its name, type, location, and value. Changing a variable's value during runtime must be accomplished by using direct memory access using **Poke** and its variants.

Property	**Description**
Name	The name of the variable
Pname	The name of the procedure the variable is declared.
Type	A value indicating the variable type (**basInt**, **basFixedStr**, etc)[1]
TypeName	A string describing the variables type (Integer, String, user-defined). For ByRef parameters or Pointer variables a Ref precedes the type name ("Ref Integer"). With constants "Const Int".
Value	The current value of the variable.
VarPtr	The address of the variable. In contrast with *Addr* with **ByRef** and **Pointer** variables, the physical address of the variable is returned.
Addr	The address of the variable. For Ref variables the address of the pointer.

Size	Returns the memory size of the variable (Integer: 4 bytes, Double: 8 bytes).
Len	Same as Size, but returns the length for a string.
IsArray	Returns True when the variable is an array.
ArrayAddr	The address of the first byte of the array.
ArraySize	The allocated memory for the array.
IndexCount	The number dimensions (See **IndexCount** in the Help).
LBound(n)	Returns the smallest available subscript for the specified dimension n of the array.
UBound(n)	Returns the largest available subscript for the specified dimension n of the array.
IsObject	Returns True when the variable is of type Object.
IsHash	Returns True when the variable is a Hash type.
Count	Returns the number of elements of the Hash variable.
IsTyped	Returns True when the variable is a user defined type.
TypeObj	Returns a **Gfa_Type** object for the variable when IsTyped is True.

Note: For **ParamArray** parameter **.Type** returns 250, **.TypeName** "ParamArray()", and **.IsArray** returns False, because a ParamArray isn't a normal array. The **.Value** property returns a variant array, so that the elements of the ParamArray are accessed using **.Value**(Idx).

The **.Type** property returns a 32-bit value indicating the type of variable. For the basic data types, these values are represented with a constant starting with 'bas'.

Constant	Value	TypeName
basEmpty	0	Empty
basNull	1	Null
basShort	2	Short (16-bit Integer (&))
basLong	3	Long (32 bit integer (%))
basInt	3	Long (32 bit integer (%))
basSingle	4	Single (4 byte floating point (!))
basDouble	5	Double (8 byte floating point (#))
basCurrency	6	Currency (@)
basDate	7	Date
basVString	8	String (in Variant)
basObject	9	Type of Object ("Command", "Font", "Collection", etc., but also "Nothing")
basError	10	Error
basBoolean	11	Boolean (Value 0 or -1 (?))
basVariant	12	Variant (used only with arrays of Variants)
basByte	17	Byte (|)
basLarge	20	Large
	243	Const Int
	244	Const Double
	245	Const Single
	246	Const Date
	247	Const Large
	248	Const Currency
	249	Const String
	250	ParamArray()
basType	251	user defined Type
basHash	252	Hash
basFixedStr	253	Fixed String

basUnknown	254	unknown
basString	255	String ($)
basArray	8192	Array

When .**Type** is **basObject** and the object refers to a late binding object, .**TypeName** returns the name of the server.

Example:

```
Dim o As Object
Set o = CreateObject("Word.Basic")
Print TypeName(o)   // returns "wordbasic".
```

Note Individual variables can be examined within a 'normal' program as well. A GLL isn't required to inspect variables, a Tron proc may display additional information also. The required GFA-BASIC 32 functions are identical to the Gfa_Var properties. For instance, to obtain a variable's type you would use **VarType**(var), to get a named description use **TypeName**(var), when a variable is an array, its characteristics are obtained using **ArrayAddr**, **Dim?**, etc.

See Also

Gfa_Vars, Gfa_Types, Gfa_Type

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_Dirty

Syntax

f? = **Gfa_Dirty**

Gfa_Dirty = f?

Description

Gfa_Dirty, **Gfa_Dirty**= gets or sets the program's saved status, indicating whether a project has changed since it was last saved. The function has the Boolean type and gets one of the following values:

True	Indicates that the program has been changed since it was created or last saved.
False	Indicates that the program has not been changed since it was last saved.

Setting the dirty status of the program indicates that the program has been changed and needs saving. The dirty status is marked by a * in the caption of the IDE window.

The following example gets the saved status and saves the program when it has been changed.

Example

```
Sub Gfa_Ex_S     // Shift+Ctrl+S
  If Gfa_Dirty Then Gfa_Save
End Sub
```

Remarks

The dirty status is set when as soon the source code text is changed, but also when the :Files section is updated or when a form in the form editor is modified. In particular, the **Gfa_Dirty** might be set when a program is compiled to create an exe, gll, or lg32. A change in any of the fields of the Compile dialog box changes the project that contains the compiler settings. So, after compiling the project must be resaved.

See Also

Gfa_Save, Gfa_Compile

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_Init and Gfa_Exit Events

Syntax

Sub **Gfa_Init**

Sub **Gfa_Exit**

Description

The Sub **Gfa_Init** is executed immediately after loading the Editor Extension. It is used to declare and initialize global variables, add menu entries to the Extra menu, and any other initialization required for the editor extension.

The Sub **Gfa_Exit** is executed just before unloading the Editor Extension. It is used to release resource that were used or allocated by the editor extension. Menu entries are automatically removed.

Remarks

Each editor extension may contain a number of event subs. When more than one GLL is loaded and each defines the same event sub they will be executed in the order that they are loaded into memory.

There are event subs to initialize and finalize the editor extension, a set event subs that is executed when a program is started and closed, two different timer events, and an event sub to handle the drag and drop functionality of the :Files tab.

See Also

Gfa_Minute, Gfa_Second, Gfa_OnRun, Gfa_OnEnd

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Sleep Command

Purpose

Sleep waits for occurrence of a message and invokes an event sub.

Syntax

Sleep [n]

n:iexp

Description

Sleep handles all pending messages for the application and switches control to the operating-environment kernel. Control returns to your application as soon as all other applications in the environment have had a chance to respond to pending events. This doesn't cause the current application to give up the focus, but it does enable background events to be processed.

The main message loop of a GFA-BASIC 32 application should always use **Sleep**, never **DoEvents. Sleep** is especially created to handle the OLE based user interface.

Sleep can be used with a parameter, which specifies the number of milliseconds to wait before returning to the application. **Sleep** 0 returns immediately to the application and behaves more like **DoEvents**.

Note: Sleep waits for a message, but in the IDE it returns after a short delay. This is due to a WM_TIMER message for the IDE itself, which allows intercepting the Ctrl-Break keys to stop the program. Without a WM_TIMER **Sleep** could wait forever, especially when all windows have been closed. A compiled program doesn't behave like this.

Example

```
OpenW 1
PrintWrap = 1
PrintScroll = 1
Do
  Sleep
  'DoEvents
  // to see the difference with DoEvents
  // remove the comment
  Print "*";
Until MouseK = 2
// Using Until Win_1 Is Nothing does not work here
// due to the inclusion of the Print "*"; line
  inside
// the loop
```

Remarks

By using **DoEvents** instead of **Sleep**, all simultaneous running programs (also server activities, printer spooler, etc.) will slow down. A loop with **DoEvents** prevents energy saving of a notebook. **DoEvents** was created only to use during long arithmetical calculation operations.

GetEvent and **Sleep** are alike. Both wait for a message before going on. **Sleep** handles all pending messages, where **GetEvent** only handles one message and uses the **Menu**() array to store messages. **Sleep** doesn't use the **Menu**() array at all.

When porting a GFA-BASIC 16 program you shouldn't use **DoEvents** or **Sleep**, but **GetEvent** or **PeekEvent**. By using **GetEvent** or **PeekEvent** you can get problems, when you use Ocx controls in your program simultaneously.

As a rule: Don’t mix the **Menu**() array handling and Ocx controls. Use **GetEvent**/**PeekEvent** only in programs, that use the **Menu()** array. A program that uses Ocxs has to use **Sleep** (and DoEvents).

See Also

DoEvents, GetEvent, PeekEvent

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

OpenW Command

Purpose

Creates a (MDI) window form.

Syntax

OpenW [options] [#]n [, x, y, w, h][, attr]

OpenW [options] **Owner** form, [#]n [, x, y, w, h][, attr]

OpenW [options] **MdiParent** [#]n [, x, y, w, h][, attr]

OpenW [options] **MdiChild Owner | Parent** form, [#]n [, x, y, w, h][, attr]

n, x, y, w, h, attr:iexp
*options:***[Tool] [Center] [Full] [Hidden] [Client3D] [Help] [Top] [Palette] [NoCaption] [NoTitle] [Fixed] [Default]**

Description

The GFA-BASIC 16 compatible command **OpenW** [#]*n, x, y, w, h, attr* opens the window with number n, where n can assume any value. When n specifies the values 0 to 31, GFA-BASIC 32 automatically provides a global **Form** variable named **Win_***n*. When the number is greater than 31 the window gets the object name **Form**(n).

NOTE: Using windows with a number greater than 31 can lead to some odd 'Access Violation' errors; sometimes these will disappear if the program is re-run, sometimes if

GFABasic is closed down and restarted and sometimes they persist. Closing down the Debug window has also been known to help; sometimes by using them with the windows numbered below 32 can cause a problem. Why this happens is currently unknown and there is no known workaround.

The upper left corner of the window is anchored at the coordinates specified with *x* and *y*. The window has the width *w* and the height *h*. By using *attr* the following window attributes can be specified:

Bit	**Value**	**Meaning**
0,1	1,2	vertical scrollbar
2,3	4,8	horizontal scrollbar
4	16	title line
5	32	close box
6	64	minimize box
7	128	maximize box
9	512	size box

attr =-1 draws all attributes.
attr = 0 draws a window with a single border and no attributes.

Without the *attr* parameter, the window gets all attributes except the scrollbars. (In GFA-BASIC 16 you would have used attr = ~15.)

OpenW creates a Form object named **Win_**n, where n is a number between 0 and 31. The GFA-BASIC 16 window management commands like **MoveW**, **SizeW**, etc. are still present, and can be used to manage the windows using pixel coordinates. When managing the **Form** using properties and methods the measurements are in twips.

Messages should be handled using event subs, like **Win_1_Activate**. For an overview of all properties, methods, and event subs see Form object.

When **OpenW** specifies a number > 31, then the properties and methods are accessed using **Form**(n).*property* and the event subs are like **Sub Form_Activate**(Index%). The window number is passed as the first argument in the sub parameter list. See also Name for more information on using window numbers beyond 31.

OpenW [option] **Owner** *name* creates a window that is to be owned by the form object *name*. The **Owner** option permits you to specify the parent form of the form being shown. When you use this option, you achieve two interesting effects: the owned form is always shown in front of its owner (parent), even if the parent has the focus, and when the parent form is closed or minimized, all forms it owns are also automatically closed or minimized. You can take advantage of this feature to create floating forms that host a toolbar, a palette of tools, a group of icons, and so on. This technique is most effective if combine it with the window state options **Fixed** and/or **Tool**/**Palette**.

The *options* argument specifies additional window state settings.

Center - centers the form.

Full - creates a maximized window, excludes **Hidden** (full windows are always visible).

Hidden - opens invisible

Client3D - sets WS_EX_CLIENTEDGE

Tool - creates a WS_EX_TOOLWINDOW

Help - includes a Help button in the window caption, excludes minimize an maximize buttons

Top - creates a topmost window

Palette - creates a WS_EX_PALETTEWINDOW

Fixed - a non-sizable window

NoCaption - no title bar

NoTitle - no title bar, alias

Default - uses Windows default values

You can create MDI parent and child windows with **OpenW** as well. To create a parent window use:

OpenW [options] **MdiParent** n (identical to **ParentW** n).

To create a MDI child window of MDI parent form *parentform*, use (**Owner** and **Parent** are identical):

OpenW [options] **MdiChild Parent** *parentform*, n

OpenW [options] **MdiChild Owner** *parentform*, n

These **OpenW** commands are identical to **ChildW** n, np)

```
OpenW MdiParent 1 , , , 300, 300
OpenW MdiChild Parent Me, 2, 0, 0
```

Example

```
OpenW # 1, 10, 10, 200, 100, -1//opens the window
  #1
Win_1.Moveable = 0
OpenW Tool Client3D Center Owner Win_1, 40
```

```
Form(40).Sizeable = 0
Do
  Sleep
Until Me Is Nothing

Sub Win_1_Activate
EndSub

Sub Form_Activate(Index%)
  If Index% = 40
    // code ..
  EndIf
EndSub
```

Remarks

The rules for windows numbered larger than 31 apply for **ChildW** as well. The number of simultaneous open windows is limited by the OS.

In contrast with **LoadForm**, the **OpenW**, **ChildW**, **ParentW**, and **Form** commands don't generate a **Load** event.

See Also

Form Object, Form, LoadForm, ParentW, ChildW, Dialog

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Ocx Command

Purpose

Creates an Ocx control in the current active form, window, or dialog.

Syntax

Ocx type name[(idx)] [[= text$] [,ID][, x, y, w, h] [, style]]

type:object typename
name:variable name (global)
idx:iexp, control array index
text$:sexp, caption (optional)
ID:iexp, identifier value for the control
x, y, w, h:iexp, position and dimension of the object
style:iexp, additional windows style constants

Description

Ocx is used to create an Ocx control in the source code, rather than in the Form Editor. **Ocx** takes at least two arguments: an Ocx type (OLE Control CoClass), and a variable name to which the object is assigned. The name represents the control in code and is a global variable of the given Ocx-type. For example, the following statement creates a Button control (Ocx type is Command) at position 10, 10 and with width = 80 and height = 24 pixels.

```
Ocx Command cmd = "Ok", 10, 10, 80, 24
```

The coordinates and size measurement are set with **OcxScale**. By default, the **Ocx** and **OcxOcx** commands use pixel coordinates. Setting **OcxScale** = 1 determines that the **Ocx** and **OcxOcx** commands use the **ScaleMode** setting of the form.

Some Ocx controls have a default position and size, either because they have a fixed position (ToolBar, StatusBar, TrayIcon) or they are invisible (ImageList, Timer, CommDlg). For instance:

```
Ocx ToolBar tb          // Aligned at top
Ocx StatusBar st        // Aligned at bottom
.SimpleText = "Ready"
Do : Sleep : Until Me Is Nothing
```

After an **Ocx** or **OcxOcx** command a hidden **With** command is active with the Ocx object just created. The **With** is valid until the next **With** or a new Ocx is created.

Once a global Ocx variable is entered in code its name is used for a kind of IntelliSense. By typing in the name followed by a dot, a context menu with possible properties and methods for that Ocx type is presented. By browsing through the list, the syntax of the property or method is displayed in the statusbar of the IDE. A selection is made by pressing ENTER, any other key closes the list. In the same way an event name can be selected. After typing 'Sub name_' a list pops up showing the possible event names for that control.

GFA-BASIC 32 supports all standard and common controls. The **Ocx** control types are: **Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, ToolBar, TrayIcon, TreeView, UpDown**.

A *control array* is a group of controls that share the same name and type. They also share the same event procedures. A control array has at least one element and can grow to as many elements as your system resources and memory permit. The maximum index you can use in a control array is 32767. Elements of the same control array have their own property settings.

Each control is referred to with the syntax *object(index)*. You specify the index of a control when you create it. The **Index** property distinguishes one element of the control array from another. When one of the controls in the array recognizes an event, a common event procedure is invoked and the value of the Index property is passed to identify which control actually triggered the event.

Example

```
Form frm1 = "GFA-Test", 10, 10, 250, 170
Ocx Command cmd(1) = "OK", 30, 100, 45, 25
cmd(1).Default = True        ' implicit With
Ocx Command cmd(2) = "Cancel", 80, 100, 45, 25
cmd(2).Cancel = True    ' explicit reference
Ocx Command cmd(3) = "But_3", 130, 100, 45, 25
Do
  Sleep
Until Me Is Nothing

Sub cmd_Click(Index%)
  Local a$ = "Command Button " & Index% & #13#10 &
    Iif(Index% <> 3, "Click OK to close main
    window", "")
  Message a$
  If Index% <> 3 Then Me.Close
EndSub
```

Remarks

Normally, GFA-BASIC 32 assigns a control a unique identifier, but when porting GFA-BASIC 16 code to GFA-BASIC 32 it might be useful to assign a custom ID-value. For instance, porting the Button command to GFA-BASIC 32 requires at least the replacement of the 'Button' keyword with **'Ocx Command** *name* ='. The ID argument may remain in the statement and used further down the program.

The order of control creation determines their Z-order and tab position. The last control created has the highest Z-order position. To bring other controls to the front when they ar overlapped by others, use the **ZOrder** method.

More complex Forms are to be created with the Form Editor, due to its finer tuning possibilities.

See Also

OcxOcx, OcxScale, OCX(), Form, Command, Option, CheckBox, RichEdit, ImageList, TreeView, ListView, Timer, Slider, Scroll, Image, Label, ProgressBar, TextBox, StatusBar, ListBox, ComboBox, Frame, CommDlg, MonthView, TabStrip, TrayIcon, Animation, UpDown

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

GetEvent Command

Purpose

Monitors menu and window events

Syntax

GetEvent

Get_Event

Description

GetEvent is implemented for compatibility reasons with GFA-BASIC 16. **GetEvent** monitors the occurrence of events in menu bars, pop-up menus, and windows. **GetEvent** waits a maximum of 0.5 seconds. The message parameters are copied to the **Menu**() array. A message is handled by responding to corresponding values in the **Menu**() array.

GetEvent is not OLE compatible, and cannot be used with Ocxs.

Example

```
Global i%
Dim m$(20)
Data Lissajous , Figure 1 , Figure 2 , Figure 3
Data Figure 4
Data End ,"", Names , Robert , Piere , Gustav
Data Emile , Hugo ,!!
i% = -1
```

```
Do
  i%++
  Read m$(i%)     //read in the menu entries
Loop Until m$(i%) = "!!"//marks the end
m$(i%) = "       "//terminates a menu
OpenW # 1
Color 8
PBox 0, 0, 639, 349
Menu m$()        //activates the menu bar
Do
  GetEvent
  Exit If MouseK = 2
  Trace MENU(1)
  Switch MENU(1)
  Case 0          // nothing happened for half a
    second
  Case 1          // keypress
  Case 2, 3       // mouse click
  Case 4 To 19    // windows message
  Case 20         // menu selection
  Case 21         // redraw
  EndSwitch
Loop
CloseW # 1

Sub Win_1_Close(Cancel?)
  // Don't allow Win_1 to be closed using the close
    button
  // only by right clicking
  Cancel? = True
EndSub
```

Remarks

The GFA-BASIC 16 functionality is fully supported when using **GetEvent** and **Menu**(). **Sleep** does not copy the message parameters to the **Menu**() array.

See Also

Sleep, DoEvents, PeekEvent, Menu()

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Debug Object

Purpose

This object is used to debug a program.

Syntax

Debug

Description

The **Debug** object sends output to the **debug output** window at run time.

Properties

BackColor | ForeColor | Left | Top | Height | Width | hWnd | OnTop | Visible

Methods

Assert | Clear | Hide | Print | Show | Trace

Example

```
Dim i As Int = 9
Debug.Show
Debug.OnTop = True
Debug "A debug message"
Debug.Trace i
Debug.Assert i > 10 ' This comment is displayed in
  the message box!
```

Remarks

By default the Debug commands will be ignored by the compiler. However, optionally, these commands may be kept in an executable (EXE) by setting the appropriate option in the compiler tab of the GFA-BASIC 32 Properties dialog box.

Debug is a shortcut for **Debug**.**Print** and can be used instead.

See Also

Assert, Trace, CallTree

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

Trace$ Variable

Purpose

Returns the command line to be executed next.

Syntax

Trace$

Description

Trace$ is a string variable which, inside the **Tron** *procedurename*, contains the command which will be executed next. **Tron** *procedurename*, specifies a subroutine which will be invoked before execution of every command. The combination of **Tron** procedurename and **Trace**$ is a very efficient way of looking for errors.

Example

```
Local mk%, mx%, my%
OpenW # 1 : Debug.Show
GraphMode R2_XORPEN
QBColor 11
Line 0, _Y - 20, _X, _Y - 20
Tron debug
Do
  Exit If Len(InKey$)
  Mouse mx%, my%, mk%
  If mk% %& 1
    If my% < _Y - 55
      Box mx%, my%, mx% + 30, my% + 30
    EndIf
  EndIf
Loop
CloseW # 1
End

Procedure debug
  If MouseK = 2 Then Debug.Print Trace$
EndProc
```

Return

Draws rectangles on the screen when the left mouse button is held down. When the right mouse button is hel down **Trace**$ shows the commands in debug output window.

Remarks

In a stand-alone program (EXE) the **Tron** command is ignored. **TraceLnr**, **ProcLnr**(p) and **ProcLineCnt**(p) are 0, **Trace$** and **SrcCode**(%) are "".

See Also

Tron, Debug, Trace, TraceLnr, TraceReg, SrcCode$, ProcLnr, ProcLineCnt

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Assert Command

Purpose

Debugging command that halts program execution if an expression is not true.

Syntax

[Debug].**Assert** boolexp

boolexp:Any valid Boolean expression that evaluates to true (nonzero) or false (0).

Description

The **Assert** command is intended for use in debugging and by default works only in the IDE and it stops the program execution if the expression evaluates to 0.
Its normal use is to check the correct value of the variables during debugging.

When the expression evaluates to False (0) a message box is displayed showing the entire line of code.

```
Assert x! <> 0              ' 0 not allowed for x!
Assert i >= 9 && i <= 27    ' i can not be between
  9 and 27 inclusive
Assert DllVersion("") = 2.2 ' Wrong GfaWin23.OCX
  runtime
```

The title of the message box is 'Assert:<ProgName>'.
The message box text is the entire **Assert** code line, including the comments, and the name of the procedure.

Remarks

Assert can not be used to display the contents of a variable.

Assert is a shortcut for **Debug.Assert**, a method of the **Debug** object like **Trace** and **Print**. By default the **Debug** object is disabled for final executables, but it can be enabled through the Compiler tab in the Properties dialog.

See Also

Debug, Trace

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Message, MsgBox & MsgBox0 Commands and Functions

Purpose

Displays a message in a dialog box, waits for the user to click a button, and returns a value indicating which button the user clicked.

Syntax

retval = **MsgBox**[**0**](prompt)

retval = **MsgBox**(prompt[, flags][, title][, helpfile, context])

MsgBox[**0**] prompt[, flags][, title][, helpfile, context][, retval]

Message prompt[, title][, flags][, retval]

[retval =] **Message**(prompt)

retval = **Message**(prompt, title, flags)

prompt, title, helpfile	*: sexp*
retval, flags, context	*: iexp*

Description

MsgBox displays a message dialog box which is owned by the current active Form while **MsgBox0** doesn't have an owner, the handle of the parent being set to 0. **MsgBox0** is particularly useful when the owner-owned relationship isn't wanted and the message box is not forced in the foreground of the form. Whether by design or error, GFA Basic only supports the **MsgBox0** function with a single parameter, but it does support the **MsgBox0** command in full.

The **MsgBox**[**0**] syntax has these arguments:

> *prompt* - String expression displayed as the message in the dialog box. The maximum length of *prompt* is approximately 1024 characters, depending on the width of the characters used. If prompt consists of more than one line, you can separate the lines using a carriage return character (Chr(13)), a linefeed character (Chr(10)), or carriage return-linefeed character combination (Chr(13) & Chr(10)) between each line.
>
> *flags* - Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, the modality of the message box and other settings which effect display and behaviour. See the Formatting & Button Options section for values. If omitted, the default value for *flags* is 0.
>
> *title* - String expression displayed in the title bar of the dialog box. If you omit *title*, the application name is placed in the title bar.
>
> *helpfile* - String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If *helpfile* is provided, *context* must also be provided.
>
> *context* - Numeric expression that identifies the Help context number assigned by the Help author to the

appropriate Help topic. If *context* is provided, *helpfile* must also be provided.

retval - This is the return value of the button selected.

When both *helpfile* and *context* are provided a Help button is added and context-sensitive Help is provided for the dialog box. However, no value is returned until one of the other buttons is clicked. In addition, when the Help button is visible, the user can press F1 to view the Help topic (WinHlp) corresponding to the context.

NOTE: With the demise of the Winhlp (.hlp) file format, *helpfile* and *context* will not work on Windows Vista (2007) onwards (unless you have installed a older version of WinHlp32.exe). To add help to a message box, see the examples in Known Issues below.

If the dialog box displays a Cancel button, pressing the ESC key has the same effect as clicking Cancel.

Message is similar to **MsgBox** and uses the same parameter values with the main difference being the omission of a link to a WinHlp help file through *helpfile* and *context*; instead, add MB_HELP to *flags* and catch the returned WM_HELP message in the parent window's **_Message** event - the pointer to the HELPINFO structure is stored in wParam. (**Note:** As with **MsgBox**, from Windows Vista onwards, trying to access a WinHlp file can cause a fatal error - see below for workarounds.)

Formatting & Button Options

Any constant marked with an asterisk (*) is not recognised as an internal value and will need to be either added as a constant with your program or used as a numerical value with the **MsgBox** function and/or command.

Button Options

The following set the array of buttons used in the message box (See example 1 in Known Issues to see how to customise the button captions):

MB_OK = $0000 - the message box contains an "OK" push button.

MB_OKCANCEL = $0001 - the message box contains two push buttons, "OK" and "Cancel".

MB_ABORTRETRYIGNORE = $0002 - message box with three buttons Abort, Retry, Ignore

MB_YESNOCANCEL = $0003 - the message box contains three push buttons "Yes", "No" and "Cancel".

MB_YESNO = $0004 - the message box contains two push buttons "Yes" and "No".

MB_RETRYCANCEL = $0005 - the message box contains two push buttons "Retry" and "Cancel".

MB_CANCELTRYCONTINUE* = $0006 - the message box contains three push buttons "Cancel", "Try Again" and "Continue" (needs to be declared).

It is possible to specify is the default (has focus) with one of the following:

MB_DEFBUTTON1 = $0000 - the first button is selected (default).

MB_DEFBUTTON2 = $0100 - the second button is selected.

MB_DEFBUTTON3 = $0200 - the third button is selected.

MB_DEFBUTTON4 = $0300 - the fourth button is selected.

If the button specified as default is not present, focus is shifted to first Button

Icon Options

The icon to be displayed in the mesage box is determined by:

MB_ICONERROR *or*
MB_ICONHAND *or*
MB_ICONSTOP = $0010 - the box contains a stop sign icon.

MB_ICONQUESTION = $0020 - the box contains a question mark icon.

MB_ICONEXCLAMATION *or*
MB_ICONWARNING = $0030 - the box contains an exclamation mark icon.

MB_ICONASTERISK *or*
MB_ICONINFORMATION= $0040 - the box contains an icon with an "i" in a circle.

Modal Settings

The following constants allow you to change to Modal status of the window:

MB_APPLMODAL = $0000 - the user must respond to a message before being able to continue working in the window which created the message.

MB_SYSTEMMODAL = $1000 - used to indicate a serious error in the program (for example "out of memory"). As a rule the program must subsequently be terminated.

MB_TASKMODAL = $2000 - same as MF_APPLMODAL. In addition all Top Level windows which belong to the current program are inactivated.

Miscellaneous Settings

Below are more settings that can be combined with those above:

MB_DEFAULT_DESKTOP_ONLY = $20000 - If the current input desktop is not the default desktop, MsgBox does not return until the user switches to the default desktop.

MB_HELP = $40000 - Add a help button to a message box. This has no effect on the **MsgBox** function if *helpfile* and *context* are not defined, and has been included in this list for use with the *MessageBox()* function which is dealt with above..

MB_RIGHT = $80000 - Right-aligns all text

MB_RTLREADING = $100000 - Prints text from right to left for languages that are written that way.

MB_SETFOREGROUND = $10000 - The message box becomes the foreground (or active) window.

MB_TOPMOST = $40000 - The message box is created with the WS_EX_TOPMOST (or system) window style.

MB_SERVICE_NOTIFICATION = $200000 - The caller is a service notifying the user of an event. The function displays a message box on the current active desktop, even if there is no user logged on to the computer.

Return values

The following are accepted return values:

IDABORT = $3 - The Abort button was pressed.

IDCANCEL = $2 - The Cancel button was pressed.

IDCONTINUE* = $11 - The Continue button was pressed (needs to be declared).

IDIGNORE = $5 - The Ignore button was pressed.

IDNO = $7 - The Nobutton was pressed.

IDOK = $10 - The OK button was pressed.

IDRETRY = $4 - The Retry button was pressed.

IDTRYAGAIN* = $11 - The Try Again button was pressed (needs to be declared).

IDYES = $6 - The Yes button was pressed.

Example

```
Local a%, b$, c$, n%
a% = MB_ABORTRETRYIGNORE
b$ = "This is a message"
```

```
c$ = "GFA-BASIC 32"
n% = MsgBox(b$, a%, c$)
MsgBox c$, , , , , n%
Message b$, "", a%, n%
```

Remarks

For an alternative style of message box, see GFA Basic's own version called Alert.

It is possible to display a message box with a check box which gives the option not to show the message again by using the *SHMessageBoxCheck()* API. A quick example is below:

```
Declare Function SHMessageBoxCheck Lib "Shlwapi"
  Alias "SHMessageBoxCheckA" (ByVal hwnd As Handle,
  ByVal Prompt As String, _
  ByVal Title As String, ByVal Flags As Long, ByVal
    DefaultID As Long, ByVal RegVal As String)
OpenW 1
Local r% = SHMessageBoxCheck(Win_1.hWnd, "Do you
  want to save this file?", "Save File?", MB_YESNO,
  IDYES, "Test")
Message "Return Value was" & r% & #13#10 & "Do you
  want to see the new registry value?", "",
  MB_YESNO, r%
If r% = IDYES
  SaveSetting
    "HKCU\software\microsoft\windows\currentversion
    \applets\regedit", "", "lastkey", Str, _
    "HKCU\Software\Microsoft\Windows\CurrentVersion
      \Explorer\DontShowMeThisDialogAgain"
  ~ShellExec("regedit.exe")
EndIf
```

If the checkbox is ticked when the message box closes, a Registry key named after *RegVal* is added to the HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer \DontShowMeThisDialogAgain key with the value 'NO'. To show the message again, either delete this key or change the value to 'YES'.

For more details on *SHMessageBoxCheck*, see MSDN.

MsgBox (and MsgBox0) uses Me as a parent and is displayed on Me’s (usually the curernt) monitor, except in GLLs, where Me is unavailable, when MsgBox0 alone should used instead.

Known Issues

As noted above, with **MsgBox**[**0**] from Windows Vista onwards, the *helpfile* and *context* parameters no longer link to the deprecated WinHlp32.exe help files. Similarly, neither does using **Message** or the internally declared *MessageBox()* with the MB_HELP flag; in fact, this should not be used as, in certain circumstances, it can cause serious errors.

There are two alternatives to this problem:

1. The first is a workaround which converts one of the other buttons into a Help button and uses the return value to branch off to the help page. This example is especially interesting as it also shows how to customise the button names.

   ```
   // Acknowledgements to Peter Heinzig
   Form F0 = , , , 400, 300 : DoEvents
   RedrawMsgBox:
   Ocx Timer Tim : Tim.Interval = 3 : Tim.Enabled
     = 1
   ```

```
If MsgBox("Tralala", MB_OKCANCEL, " ") =
  IDCANCEL // Cancel is now Help
  // Call your helpfile/page.
  GoTo RedrawMsgBox
EndIf
F0.Close

Sub Tim_Timer // Use to Change text in
  messagebox
  ~SendDlgItemMessage(GetActiveWindow(),
    IDCANCEL, WM_SETTEXT, 0, "Help") // "Cancel"
    => "Help"
  Set Tim = Nothing // Cancel Timer as task done
EndSub
```

2. The second method is longer and uses the *MessageBoxIndirect()* API:

```
Type MSGBOXPARAMS
  - Long Size, Owner, hInstance
  - Long Text, Caption, Style, Icon,
    ContextHelpId
  - Long MsgBoxCallBack, LanguageId
EndType
OpenW 1 : Debug.Show
Local mbp As MSGBOXPARAMS, a$ = "This is a
  trial MessageBox", b$ = "Trial Help"
mbp.Size = SizeOf(MSGBOXPARAMS)
mbp.Owner = Win_1.hWnd
mbp.hInstance = Null
mbp.Text = V:a$
mbp.Caption = V:b$
mbp.Style = MB_OK | MB_HELP
mbp.Icon = Null
mbp.ContextHelpId = 12
mbp.MsgBoxCallBack = ProcAddr(HelpRoutine)
Print MessageBoxIndirect(mbp)
```

```
Do : Sleep : Until Win_1 Is Nothing

Procedure HelpRoutine(helpptr%)
  Local hi As HELPINFO
  MemCpy V:hi, helpptr%, SizeOf(HELPINFO)
  Debug hi.ContextId
  Type HELPINFO
    - Long Size, ContextType, CtrlId
    - Long ItemHandle, ContextId
    MousePos As POINT
  EndType
  Type POINT
    - Long x, y
  EndType
EndProcedure
```

For more information on the possible values in the MSGBOXPARAMS structure, see MSDN.

For more information on linking to HTML Help Files, see Accessing HTML Help Files.

See Also

Alert

{Created by Sjouke Hamstra; Last updated: 04/04/2018 by James Gaite}

Gfa_StatusText

Return or set the status bar text.

Syntax

$ = **Gfa_StatusText**

Gfa_StatusText [= text$]

Description

Gfa_StatusText [=] returns or sets the text of the status bar of the IDE. The text is not permanent, because it is overwritten by GFA-BASIC 32 when it displays information like menu item description, OCX properties and methods, import descriptions, etc.

Example

```
Gfa_StatusText = "Ready"
```

See Also

Gfa_hWnd, Gfa_hWndEd

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_KeyGet

Syntax

vkkeycode% = **Gfa_KeyGet**

Description

Returns the virtual key code for a pressed key. This function could be used for various purposes, but can be invoked not before an editor extension is invoked. It is safely implemented as a *PeekMessage* loop filtering keyboard messages. Once executed the **Gfa_KeyGet** function exits and returns 0 after a time out of 60 seconds when no keyboard message has arrived. It also returns 0 when one of the mouse buttons is clicked, a menu is selected, Alt is pressed, or when a WM_APPACTIVATE is received.

Gfa_KeyGet ignores the shift state of the Shift keys, it returns the codes 8, 9, 13, 27, and greater than 31 that are in the low order word of the wParam of the WM_KEYDOWN message.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

InputBox Function

Purpose

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns the contents of the text box.

Syntax

string = **InputBox**(prompt[, title][, default][, x][, y][, helpfile, context])

prompt, title, default	*: sexp*
x, y, context	*: iexp*
helpfile	*: path to .hlp help file*

Description

prompt: String expression displayed as the message in the dialog box. The maximum length of prompt is approximately 1024 characters, depending on the width of the characters used. If prompt consists of more than one line, you can separate the lines using a carriage return character (Chr(13)), a linefeed character (Chr(10)), or carriage return-linefeed character combination (Chr(13) & Chr(10)) between each line.

title: String expression displayed in the title bar of the dialog box. If you omit title, the application name (App.Name) is placed in the title bar.

default: String expression displayed in the text box as the default response if no other input is provided. If you omit default, the text box is displayed empty.

x: Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If x is omitted, the dialog box is horizontally centered.

y: Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog box from the top of the screen. If y is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen.

helpfile: String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If helpfile is provided, context must also be provided.

context: Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If context is provided, helpfile must also be provided.

When both helpfile and context are supplied, a Help button is automatically added to the dialog box. **NOTE** The help button on an **InputBox** will attempt to open WinHlp32.exe (used for .hlp files); it will not work with HTMLHelp files. Unfortunately, as InputBox is an internally created Dialog box rather than a Windows API, the only workaround is to create a custom Dialog Box and direct the call to the help file in a similar manner as shown in this example.

If the user clicks OK or presses ENTER, the InputBox function returns whatever is in the text box. If the user clicks Cancel, the function returns a zero-length string ("").

Example

```
OpenW 1
```

```
Local x%, value$
value$ = InputBox("Hallo", "Title", "Mr.")
Print value$
```

Remarks

In contrast to VB the coordinates will be corrected automatically by GFA-BASIC 32, so that the **InputBox** remains on the screen.

Input Boxes are useful inside LG32 Libraries as, unlike OCX objects, their events can be handled internally.

InputBox uses Me as a parent and is displayed on Me's (usually the curernt) monitor, except in GLLs, where Me is unavailable, when MsgBox0 should used instead.

See Also

Prompt, Input

{Created by Sjouke Hamstra; Last updated: 04/04/2018 by James Gaite}

Prompt Command

Purpose

Displays an input Dialog Box

Syntax

Prompt title$, message$, strvar$

Description

Prompt will provide the user with a "standard" dialog box that has an input-field in it. Basically this can be used instead of **Input** or **Form Input** to prompt the user for input, because Input is not very suited for Message-based multi-tasking systems like MS-Windows.

Example

```
Local a$ = "Anonymous"
Prompt "A Prompt example", "Do you want to give
  your name ?", a$
Print a$
```

This example prompts the user with a Dialog-box, asking to type in your name. The default text in the edit field will be "Anonymous".

After response of the user, the string A$ will be filled with the text of the edit field at the moment of exiting the Dialog.

Remarks

Basically, the same can be done using Dialog and EditText statements. Of course, the Prompt command is easier to use and provides a kind of standard-Dialog for user input.

Prompt uses Me as a parent and is displayed on Me's (usually the curernt) monitor, except in GLLs, where Me is unavailable, when MsgBox0 should used instead.

See Also

Dialog, Form Input

{Created by Sjouke Hamstra; Last updated: 04/04/2018 by James Gaite}

PopUp Menus

Purpose

Creates a pop-up menu.

Syntax

r = **PopUp**(entries$, x, y, i)

PopUp entries$, x, y, i, (OUT) r

entries$	*: string*
r, x, y, i	*: integer*

Description

As opposed to Menu Bars, pop-up menus are not permanent, have one main column and can be deployed anywhere within a form - the best example of a pop-up menu is one that is produced when you use the right mouse button to click on a certain object to get further options.

Similar to Menu Bars, pop-up menus can be created either through a GB32 command - in this instance **PopUp** - or through calling Windows' internal APIs, which has the added advantages of allowing sub-menus to be created and custom ID numbers to be assigned to menu items; unlike Menu Bars, due to its brief existence and the structure of the **PopUp** command/function, a pop-up menu can not be created by using both methods.

Creating Pop-Up Menus using PopUp Show

Creating Pop-Up Menus using APIs Show

See Also

Menus

{Created by Sjouke Hamstra; Last updated: 20/12/2015 by James Gaite}

Gfa_Exit and Gfa_DoExit Commands

Quit IDE.

Syntax

Gfa_Exit

Gfa_DoExit

Description

Gfa_Exit executes the File | Exit menu item to close GFA-BASIC 32. When the current project has the **Gfa_Dirty** status the project can be saved first.

Gfa_DoExit closes GFA-BASIC 32 without checking the **Gfa_Dirty** status. There is no correspondence menu item for this command.

Example

```
Sub Gfa_Ex_X
  Gfa_DoExit
EndSub
```

See Also

Gfa_Dirty

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_OnRun and Gfa_OnEnd Events

Syntax

Sub **Gfa_OnRun**

Sub **Gfa_OnEnd**

Description

The Sub **Gfa_OnRun** is called directly before the start of a program. This event could be used to save the latest changes before running. This event might also be used to start the execution of a **Gfa_Tron** proc for debugging purposes.

The Sub **Gfa_OnEnd** is called directly after the end of a program. **Gfa_OnRun** and **Gfa_OnEnd** subs could be used to minimize the IDE when a program is started.

Example

```
// Use Gfa_OnRun to backup file in system's
 temporary directory.

Sub Gfa_OnRun   //Backup before program start
  Debug "Starting Program"
  If Gfa_Dirty Then Gfa_SaveFile TempDir &
   "run.g32"
  ShowW Gfa_hWnd, SW_MINIMIZE
End Sub

Sub Gfa_OnEnd
```

```
  ShowW Gfa_hWnd, SW_RESTORE
End Sub
```

See Also

Gfa_Run, Gfa_Init, Gfa_Exit, Gfa_Minute, Gfa_Second

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_Minute and Gfa_Second Events

Syntax

Sub **Gfa_Minute**

Sub **Gfa_Second**

Description

The **Gfa_Minute** sub is called every minute (approximately), except if the program is running. Although it isn’t advised to perform complex and lengthy actions because this could slow down the editor, there is quite some room here to create useful extensions. The GFA-BASIC 32 editor uses the same timer interrupt to clock in the status bar.

The **Gfa_Second** sub is called every second (approximately), except if the program is running. Although it isn’t advised to perform complex and lengthy actions because this could slow down the editor, there is quite some room here to create useful extensions. The GFA-BASIC 32 editor uses the same timer interrupt to update the title of the IDE when the dirty status of program has been changed.

Example 1

```
// Changing the timer interrupts
Global Const tMinuteId = $14D
Global Const tSecondId = $14E
```

```
~KillTimer(Gfa_hWnd, tSecondId)        'Set the
  Gfa_Second timer to
~SetTimer(Gfa_hWnd, tSecondId, 500, 0) '500
  milliseconds, rather than 1000ms
```

Example 2

```
//Show current procedure in the status bar each
  second.

Sub Gfa_Second
  Gfa_StatusText = Gfa_Proc
End Sub
```

The second example of **Gfa_Second** is changed somewhat, and behaves more reservedly. Thus the **Gfa_StatusText** is changed only when the current procedure’s top line changes.

Example 3

```
// Display current procedure in status bar.

Sub Gfa_Second
  Static Int procline
  If procline != Gfa_ProcLine
    procline = Gfa_ProcLine
    Gfa_StatusText = Gfa_Proc
  EndIf
EndSub
```

See Also

Gfa_OnRun, Gfa_OnEnd, Gfa_Init, Gfa_Exit

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_OnDropInl Event

Syntax

Sub **Gfa_OnDropInl**(ParamArray p())

Description

When the Sub **Gfa_OnDropInl**(ParamArray p()) exists the ':Files' tab in the sidebar will become a drag and drop window. When one or more files are dragged from the Explorer to the ':Files' window the **Gfa_OnDropInl** sub is invoked. The ParamArray p() contains the strings with filenames that are dropped in the operation.

Example

Add resources using drag 'n drop.

When a GLL contains the **Gfa_OnDropInl** sub, the drag and drop facility of the :Files tab is enabled. The **Gfa_OnDropInl** takes one parameter: a ParamArray containing the list of files to add.

```
Sub Gfa_OnDropInl(ParamArray p())
  Local Int i
  For i = LBound(p) To UBound(p)
    dropfile p(i)
  Next
End Sub

Sub dropfile(f$)
  Debug "Dropped file " + f
  Local a$, i%
  Try
    a = f
    If(FileLen(f) > 8192)
      If MsgBox("File length " & f & " =" &
        FileLen(f$) & "Bytes"#10"Copy anyway?", _
        MB_YESNO) == IDNO Then Exit Sub
    EndIf
    i% = RInStr(f$, "\")
    If i%
      a = Mid(f, i + 1)
      i% = RInStr(a, ".")
      If i > 1 Then a = Left(a, i - 1)
      a = ":" & a
      If Exist(a)
        //iiiFile is exiting
        i = MsgBox("InlFile " & a & " exists" #10
          "Overwrite " & f & "?", MB_YESNOCANCEL)
        If i = IDCANCEL Then Exit Sub // nothing to
          do
        If i = IDYES Then Gfa_CopyFile "", a :
          Gfa_CopyFile f, a : Exit Sub
        For i = 0 To 99
          If !Exist(a & Dec(i))
            Gfa_CopyFile f, a & Dec(i)
            Exit Sub
          EndIf
        Next
        MsgBox "Too much copies."
        Exit Sub
      EndIf
      Gfa_CopyFile f, a
    EndIf
  Catch
    MsgBox "Error creating a copy of " & f
  EndCatch
End Sub
```

The *dropfile* sub is called for each file in the ParamArray. First the size of the file is tested, because including resources larger then 8192 bytes might not be advisable. A confirmation is asked, therefore. Then the filename is obtained from the full path name and section without the extension is used as the ':File' name. When the ':File' exists in the inline section, you'll be asked to delete it first. If OK the resource is deleted from memory and the new file is added.

Remarks

See Also

Gfa_CopyFile, Gfa_InlFileName

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Dialog Command

Purpose

creates a Form or dialog box in a GLL using a dialog box syntax.

Syntax

Dialog hd%,x%,y%,w%,h%,tit$ [,flag% [,height%,font$]]

EndDialog

hd%,x%,y%,w%,h%,flag%,height%:integer expression
tit$, font$:sexp

Description

Dialog boxes are used for interaction between the program and the user. In contrast with GFA-BASIC 16 dialog boxes created with the **Dialog** command are OCX Forms. As such it is easier to create a dialog as a Form using the Form editor.

The **Dialog** command is a hold over from GFA-BASIC 16. The way they are used in GFA-BASIC 32 is the same as in GFA-BASIC 16. In particular, when a dialog box is created in a GLL the syntax and message handling is the same.

Formally, a **Dialog** structure has the following layout:

Dialog
Dialog control elements
EndDialog

The **Dialog** structure control elements are specified within the **Dialog**-**EndDialog** definition. The **Dialog** structure header marks the Dialog definition. It is followed by six parameters:

hd%:	Dialog structure number (0 to 31)
x%,y%:	X, Y coordinates of upper left corner of Dialog box
w%:	Dialog box width in pixels
h%:	Dialog box height in pixels
tit$:	Dialog structure title

Optionally three other parameters can be defined:

flags%:	WS_Style flags to be used by the Dialog
height%:	Font-height (normally negative)
font$:	Typeface name of the font

flags% can be a combination (binary Or) of the following values:

WS_BORDER ($00800000)	window with a border
WS_CAPTION ($00C00000)	creates a window with a title. To make a system menu visible in such a window the WS_CAPTION and WS_POPUPWINDOW style elements must be combined.
WS_CHILD ($40000000)	a window with child windows
WS_CHILDWINDOW	a child window
WS_CLIPCHILDREN ($02000000)	clips all window output to the area outside of a child window.
WS_CLIPSIBLINGS	clips all window output within

($04000000)	a child window to its client area.
WS_DISABLED ($08000000)	a window, which is initially inactive.
WS_DGLFRAME ($00400000)	a window with a double border but without a title.
WS_GROUP ($00020000)	marks the first control element within a group of control elements (used only in dialog boxes).
WS_HSCROLL ($00100000)	a window with a horizontal scroll bar.
WS_ICONIC ($20000000)	a window which is initially displayed as an icon.
WS_MAXIMIZE ($01000000)	a window with maximum dimensions
WS_MAXIMIZEBOX ($00010000)	a window with a maximize box.
WS_MINIMIZE ($20000000)	a window with minimal dimensions.
WS_MINIMIZEBOX ($00020000)	a window with a minimize box.
WS_OVERLAPPED ($00000000)	an overlapping window. The window contains a border and a title. The client area overlaps with window border and title.
WS_OVERLAPPEDWINDOW (0xCF0000)	an overlapping window with following style elements: WS_OVERLAPPED WS_CAPTION WS_SYSMENU WS_THICKFRAME

	WS_MINIMIZEBOX WS_MAXIMIZEBOX
WS_POPUP ($80000000)	a popup window. Such window can't have the WS_CHILD attribute.
WS_POPUPWINDOW (0x80880000)	a popup window with following style elements: WS_BORDER WS_POPUP WS_SYSMENU
WS_SYSMENU ($00080000)	a window with a system menu in the title bar. Used only in windows with a title bar.
WS_TABSTOP ($00010000)	a window with a number of control elements which the user can arrive at by tapping the tab key. Used only in dialog boxes.
WS_THICKFRAME ($00040000)	a window with a thick border which is used to "size" the window.
WS_VISIBLE ($10000000)	a window which is initially visible, i.e. is displayed as a top window.
WS_VSCROLL ($00200000)	a window with a vertical scroll bar.
DS_LOCALEDIT	Specifies that edit controls in the Dialog box will use memory in the application's data segment. By default, all edit controls in Dialog boxes use memory outside the application's data segment.

	This feature can be suppressed by adding the DS_LOCALEDIT flag to the STYLE command for the Dialog box. If this flag is not used, EM_GETHANDLE and EM_SETHANDLE messages must not be used since the storage for the control is not in the application's data segment. This feature does not affect edit controls created outside of Dialog boxes.
DS_MODALFRAME	Creates a Dialog box with a modal Dialog box frame that can be combined with a title bar and System menu by specifying the WS_CAPTION and WS_SYSMENU styles.
DS_NOIDLEMSG	Suppresses WM_ENTERIDLE messages that Windows would otherwise send to the owner of the Dialog box while the Dialog box is displayed.
DS_SYSMODAL	Creates a system-modal Dialog box.

A **Dialog** is a **Form** object and does not have the WS_POPUP style as a normal API dialog box. As any other Form a **Dialog** box is a WS_OVERLAPPED window. It is simply another way to

A program can define several **Dialog** structures, which are referred to by their **Dialog** number. After a dialog structure has been defined it can be displayed by using the **ShowDialog** command, where only the number of the dialog structure must be specified.

A dialog box is a **Form** object, unless used in a GLL. In a GLL the dialog box is plain API dialog box that is to be filled with plain controls. You can still use plain controls in a dialog, but you cannot respond to event subs.

Because the dialog is a **Form**, they need an object name. The **Dialog** command accepts a unique number in the range from 0 to 31. The dialog box with number #0 is named **Dlg_0**, the dialog box with #1 is called **Dlg_1**, etc. Properties and methods are invoked as Dlg_1.Property and Dlg_1.Method. The events for the dialog box are the same as for a form and have the form of Dlg_n_*event.* For instance, the event sub to handle posted messages, which are retrieved from the message queue:

```
Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  ' Code
EndSub
```

As with any Form, controls may be created the API way or the OLE way, using the **OCX** command. An advantage of using OCX controls is the way notification messages from the control are handled, because messages from the OCX controls are handled in OCX event subs.

OCX Controls

OCX type name

OCX type name = "Caption", x, y, w, h

OCX type name = "Caption", id, x, y, w, h

OCX type name = "Caption", id, x, y, w, h, style

type	Name of the GFA-BASIC 32 OCX type: **Command, Option, CheckBox, RichEdit, ImageList, Label, ListBox, TreeView, ListView, TextBox, Image, Timer, Scroll, Slider, ProgressBar, ToolBar, StatusBar, ComboBox, Frame, TabStrip, Animation, UpDown, Form**.
name	name of the global variable for the OCX. Defines the names for the event subs: *name_event*
Caption	Specifies text that is displayed with the control.
ID%	Optional. Specifies the control identifier. (0through 65,535). Normally, GFA-BASIC 32 assigns OCX controls an identifier, but for GFA-BASIC 16 programs it may be handy to keep the identifier value.
x%, y%	Specifies the x- and y-coordinate of the left top side of the control relative to the left top side of the dialog box. The coordinate is in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width%	Specifies the width of the control.
height%	Specifies the height of the control.
style%	Optional. Specifies the control styles. Use the bitwise OR (|) operator to combine styles.

Plain Controls

The dialog box can also contain plain controls. In a GLL only plain controls can be used. For instance, to create a simple left justified static text control:

LText text$, ID%, x%, y%, width%, height% [,style%]

All control statements use the same syntax:

CtrlName *text$, ID%, x%, y%, width%, height% [,style%]*

CtrlName	Name of the GFA-BASIC 32 control statement: **LText, RText, CText, Icon, PushButton, DefPushButton, CheckBox, AutoCheckBox, RadioButton, AutoRadioButton, ListBox, ComboBox, EditText, Scrollbar, AnimateCtrl, TabCtrl, HeaderCtrl, ListViewCtrl, TreeViewCtrl, ProgressCtrl, TrackBarCtrl, StatusCtrl, ToolBarCtrl, UpDownCtrl. RichEditCtrl**.
text$	Specifies text that is displayed with the control.
ID%	Specifies the control identifier. (0through 65,535)
x%, y%	Specifies the x- and y-coordinate of the left top side of the control relative to the left top side of the dialog box. The coordinate is in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width%	Specifies the width of the control.

height%	Specifies the height of the control.
style%	Specifies the control styles. Use the bitwise OR (|) operator to combine styles.

Note - There is no **Static** control command, **Static** is used to declare static local variables. Use the general **Control** statement instead.

In a normal program the messages from plain controls are handled in the parent's event sub **_Message**() (for posted messages) or **_MessageProc**() (for all messages). In a GLL the messages are handled in the **Gfa_CB**() callback sub.

The Control statement

The **Control** statement is used to create a plain control. In a GLL try to avoid the general **Control** statement to create a child window, these controls use the system font, rather than the DEFAULT_ GUI_FONT.

Control text$,ID%,class$,style%,x%,y%,w%,h%,

Control creates a program defined control window with width w% and height h% at coordinates specified in x% and y%. The window shows the text specified in text$ and can be referred to with the value specified in ID%.
class$ specifies the class of the control elements which the control window can assign.

Example

```
Call demodialog() // dialog structure
// activate DefPushButton
~SetFocus(DlgItem(1, 103))
Do
  Sleep // to wait of a message
```

```
Until MouseK = 2 // till left mouse key pressed
CloseDialog # 1 // close dialog

Procedure demodialog() // to build the dialog
  Local i%, a%
  Local dlgf&, s%, style1%, style2%
  Local style3%, style4%, style5%
  Local style6%, style7%, style8%, v%
  DlgBase Pixel // dialog in pixels
  Dialog # 1, 10, 100, 600, 360, "Demo Dialog"
    DlgBase Unit
    // rest of it in UNITS (1/4 sign width, 1/8
      sign height
    style1% = WS_TABSTOP
    style2% = BS_DEFPUSHBUTTON | WS_TABSTOP
    style3% = BS_GROUPBOX | WS_TABSTOP
    style4% = BS_AUTORADIOBUTTON | WS_TABSTOP
    style5% = BS_AUTOCHECKBOX | WS_TABSTOP
    style6% = ES_UPPERCASE | WS_BORDER | _
      WS_TABSTOP
    style7% = LBS_NOTIFY | LBS_SORT | _
      LBS_STANDARD _
      | WS_BORDER | WS_VSCROLL
    style8% = CBS_DROPDOWN | CBS_SORT | _
      CBS_HASSTRINGS | WS_VSCROLL
    // type title / contents Id x y w h style
    PushButton "Pushbutton 1", 100, 12, 14, 72, 14,

      _
      style1%
    PushButton "Pushbutton 2", 101, 12, 32, 72, 14,

      _
      style1%
    PushButton "Pushbutton 3", 102, 12, 50, 72, 14,

      _
      style1%
    DefPushButton "DefPushbutton", 103, 12, 68, 72,
      14, _
    style2%
   ScrollBar "", 104, 0, 143, 283, 9, SBS_HORZ
   ScrollBar "", 105, 283, 0, 9, 152, SBS_VERT
   GroupBox "Radiobuttons", 106, 89, 14, 56, 53,
    style3%
   RadioButton "Radio 1", 107, 93, 25, 39, 12,
    style4%
   RadioButton "Radio 2", 108, 93, 36, 39, 12,
    style4%
   RadioButton "Radio 3", 109, 93, 47, 39, 12,
    style4%
   CheckBox "Checkbox 1", 110, 17, 94, 61, 12,
    style5%
   CheckBox "Checkbox 2", 111, 17, 107, 61, 12,
    style5%
   CheckBox "AutoCheckbox", 112, 17, 120, 61, 12,

    _
    style5%
   EditText "", 113, 89, 94, 59, 12, style6%
   EditText "", 114, 89, 107, 59, 12, style6%
   EditText "", 115, 89, 120, 59, 12, style6%
   ListBox "", 116, 154, 16, 64, 113, style7%
EndDialog
// Fill List- and Combobox
For i% = 1 To 50
   s% = LB_ADDSTRING
   v% = Rand(100)
   ~SendMessage(DlgItem(1, 116), s%, 0, Str$(v%,
    2) _
    + ".String")
   s% = CB_ADDSTRING
   v% = Rand(500)
   ~SendMessage(DlgItem(1, 117), s%, 0, Str$(v%,
    4) _
    + ".String")
Next i%
// Init Scrollbars
```

```
  ~SetScrollRange(DlgItem(1, 104), SB_CTL, 0, 200,
   1)
  ~SetScrollPos(DlgItem(1, 104), SB_CTL, 100, 1)
  ~SetScrollRange(DlgItem(1, 105), SB_CTL, 0, 200,
   1)
  ~SetScrollPos(DlgItem(1, 105), SB_CTL, 100, 1)
  ShowDialog # 1
EndProc
```

Remarks

Note - GFA-BASIC 32 also provides keywords like ProgressBar, Toolbar, Header, etc. These keywords are not statements to create controls, but they are OCX *types*. As such these keywords are used to declare variables or to create OCX controls. For instance:

```
Dim pb1 As ProgressBar     ' declare a variable pb
Ocx ProgressBar pb1        ' create OCX & declare
  global variable pb1
```

Note - The **Dialog** command is useful for converting GFA-BASIC 16 programs and in GLL extensions. In a normal program use **Form** instead.

OCX types are not allowed in a GLL.

See Also

ShowDialog, CloseDialog, PushButton, DefPushButton, EditText, CText, RText, LText, Static, ScrollBar, ComboBox, ListBox

{Created by Sjouke Hamstra; Last updated: 30/09/2014 by James Gaite}

ShowDialog Command

Purpose

Displays a Dialog structure on the screen.

Syntax

ShowDialog id%

id%:integer expression

Description

ShowDialog displays a dialog structure created with Dialog....EndDialog on the screen. id% is the ID number which you used during the creation to identify the dialog.

Example

See Dialog

Remarks

If the command **EndDialog** wasn't used the ShowDialog contains the structure of the current Dialog.

See Also

Dialog, EndDialog, CloseDialog

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

CloseDialog Command

Purpose

Deletes a Dialog box shown with **ShowDialog** from the screen.

Syntax

CloseDialog n

Description

n is a value between 0 and 31.

Example

```
Dlg 3D On
Dialog # 1, 10, 10, 200, 100, "Trial Dialog",
  WS_SYSMENU
  PushButton "Close", 11, 50, 20, 80, 25, 0
EndDialog
ShowDialog # 1
Do : Sleep : Until Me Is Nothing
Dlg 3D Off

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  Switch Mess
  Case WM_COMMAND
    Trace wParam
    If wParam = 11 Then CloseDialog # 1
  EndSwitch
EndSub
```

See Also

Dialog, ShowDialog

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

DlgBase Command

Purpose

scales the measurements of Dialog boxes and Dialog Controls.

Syntax

DlgBase Pixel

DlgBase Unit

DlgBase InSide

DlgBase OutSide

DlgBase Font font$

DlgBase Bold

DlgBase Bold Off

Description

When creating Dialog boxes with the GFA-BASIC structure **Dialog...EndDialog**, their position, width and height can be specified either in pixels (**DlgBase Pixel**) or in Dialog Units (**DlgBase Unit**). Units are 1/4 character wide and 1/8 character high. If a Dialog has been created using the MS-Windows SDK editor, you should always scale using **DlgBase Unit** to avoid unnecessary calculations.

DlgBase InSide and **DlgBase OutSide** determine the meaning of the rectangle coordinates specified with the

Dialog #n, x, y, w, h command. **DlgBase InSide** forces the dialog to use the rectangle as the client size. The outside coordinates are calculated using the Windows system settings. This way the client area of the dialog is the same on each Windows system. **DlgBase OutSide** switches back to the default setting: the coordinates are the dimensions of the bounding rectangle of the dialog box.

DlgBase Font font$ and **DlgBase Bold** affect the font used in the controls in the dialog box. The format of font$ is according the format in **_font$**. **DlgBase Bold** is only used when the Dialog command includes a font description, for instance

Dialog #n, x, y, w, h, "Title", style, font_height, "Fontname".

The font activated this way is a normal (not bold) version. The command **DlgBase Bold** forces the use of a bold font.

The different parameters can be combined, such as:

DlgBase Inside, **Font** "Ms Sans Serif,-8, 8"

Example

```
DlgBase OutSide, Font "Arial,-12,7" // This works
  if …
// you remove ',-12,"ARIAL"' in the next line
Dialog # 1, 50, 50, 200, 110, "DlgBase Outside",
  $80 ', -12, "ARIAL"
  LText "This should be bold!", 3, 32, 16, 350, 16,
    $0
  PushButton "Close", IDOK,  55, 45, 80, 20
EndDialog
Dlg_1.AutoClose = 1
Dlg Fill 1, SysCol(COLOR_BTNFACE)
```

```
ShowDialog # 1
DlgBase InSide
Dialog # 2, 260, 50, 200, 110, "DlgBase inside",
  $80, -12, "ARIAL"
  LText "This is normal", 3, 32, 16, 350, 16, $0
  PushButton "Close", IDOK,  55, 45, 80, 20
EndDialog
Dlg_2.AutoClose = 1
Dlg Fill 2, SysCol(COLOR_BTNFACE)
ShowDialog # 2
Repeat
  Sleep
Until Me Is Nothing

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  If Mess% = WM_COMMAND And wParam% = IDOK Then
    CloseDialog # 1 : CloseDialog # 2
EndSub

Sub Dlg_2_Message(hWnd%, Mess%, wParam%, lParam%)
  If Mess% = WM_COMMAND And wParam% = IDOK Then
    CloseDialog # 1 : CloseDialog # 2
EndSub
```

Remarks

Bug - **DlgBase Bold** doesn't work properly. Instead use **DlgBase Font** with a bold parameter. Defining a font this way excludes the use of font parameters in the **Dialog** command.

These commands are only implemented for compatibility with GFA-BASIC 16 bit. They are however, useful in dialog boxes in a GLL.

See Also

Dialog, Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

LText Control

Purpose

Creates a left justified-text static control in the current active form, window, or dialog.

Syntax

LText text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

LText creates a rectangle with width w% and height h%, whose upper left corner is at the coordinates specified in x% and y%. The text specified in text$ is displayed in this rectangle left justified. WS_TABSTOP and WS_GROUP are available as style elements.

style Specifies the control styles. This value can be any combination of the following styles: SS_LEFT, WS_TABSTOP, and WS_GROUP. If you do not specify a style, the default style is SS_LEFT | WS_GROUP.

Creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's _MessageProc sub.

Example

```
LText "Filename", 101, 10, 10, 100, 100
Do : Sleep : Until Me Is Nothing
```

creates a left-text control that is labeled 'Filename'.

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

RText Control

Purpose

Creates a right justified text static control in the current active form, window, or dialog.

Syntax

RText text$, id%, x, y, width, height[, style%]

text$:control text
id%:control identifier
x,y,b,h:iexp
style%:the control styles

Description

RText creates a rectangle with width w% and height h%, whose upper left corner is at the coordinates specified in x% and y%. The text specified in text$ is displayed in this rectangle right justified. ID% is an integer value used to refer to (inquire about) an element. WS_TABSTOP and WS_GROUP are available as style elements.

style Specifies the control styles. This value can be any combination of the following styles: SS_RIGHT, WS_TABSTOP, and WS_GROUP. If you do not specify a style, the default style is SS_RIGHT | WS_GROUP.

Creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's _MessageProc sub.

Example

```
RText "Filename", 101, 10, 10, 100, 100
```

creates a right justified text control that is labeled 'Filename'.

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

CText Control

Purpose

Creates a centered-text static control in the current active form, window, or dialog.

Syntax

CText text$, id%, x, y, width, height[, style%]

text$:control text
id%:control identifier
x,y,b,h:iexp
style%:the control styles

Description

The control is a simple rectangle displaying the given text centered in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

style Specifies the control styles. This value can be any combination of the following styles: SS_CENTER, WS_TABSTOP, and WS_GROUP. If you do not specify a style, the default style is SS_CENTER | WS_GROUP.

Creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's _MessageProc sub.

Example

```
CText "Filename", 101, 10, 10, 100, 100
```

creates a centered-text control that is labeled 'Filename'.

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Icon, SmallIcon Properties (ListItem, Form, TrayIcon)

Purpose

Returns or sets the index value of an icon or small icon associated with a **ListItem** object in an **ImageList** control.

Returns or sets the icon for a **Form's** title bar or a **TrayIcon's** taskbar icon.

Syntax

Listitem.**Icon** [= *index%*]

ListItem.**SmallIcon** [= *index%*]

Form.**Icon** [= *pic*]

TrayIcon.**Icon** [= *pic*]

Form.**SmallIcon** [= *pic*]

ListItem:ListItem Object
Form:Form Object
index:iexp or sexp
pic:Picture Object

Description

For a **ListItem** object *index* specifies an integer that identifies an icon or small icon in an associated **ImageList** control. An **ImageList** control is associated by setting the ListView's **Icons** or **SmallIcons** property.

For a **Form** the **Icon** is 32x32 pixel bitmap and **SmallIcon** a 16x16 bitmap. The small icon is displayed in the title bar of the Form and the large icon when <Alt-Tab> is pressed. Only one needs to be set. See **LoadPicture** on how to load an icon file. **Icon** and **SmallIcon** can be set at design time, as well at run time.

For the **TrayIcon** property **Icon** the picture must be an ICO-picture. The taskbar supports 16x16 icons only. When the **Icon** property is assigned a picture at design time a 32x32 icon is loaded. The icon is then shrinked when placed in the taskbar. It is advised to load a 16x16 icon in code using:

```
Set tic1.Icon = LoadPicture(":ticSym", 16, 16, 16)
```

Example

```
// Pre-save required icon
Dim p As Picture
Set p = CreatePicture(LoadIcon(Null,
  IDI_APPLICATION), False)
SavePicture p, App.Path & "\app.ico"
// The example
OpenW 1
Print "Press any key to change the Window icon"
While InKey = "" : Wend
Cls
Set p = Win_1.Icon
Win_1.Icon = LoadPicture(App.Path & "\app.ico")
Print "Press any key to change it back"
While InKey <> "" : Wend : While InKey = "" : Wend
Cls
Win_1.SmallIcon = p
Print "Please close window to end example"
Do : Sleep : Until Win_1 Is Nothing
```

See Also

Form, ListItem, LoadPicture, TrayIcon

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

PushButton Control

Purpose

Creates a pushbutton with width, height and upper left corner is at the coordinates specified

Syntax

PushButton text$,ID%,x%,y%,w%,h%[,style%]

text$:control text
id%:control identifier
x,y,w,h:iexp
style%:the control styles

Description

A **PushButton** contains the text specified in text$ within the rectangle. When a mouse click occurs within a PushButton, it sends a message to its window. PUSHBUTTONs can contain the WS_TABSTOP, WS_DISABLED, and WS_GROUP style elements.

BS_PUSHBUTTON ($0000) - A button defined in this way sends a message to the parent window when clicked on.

BS_DEFPUSHBUTTON ($0001) - defines a button which is preselected as default (thick border; selection by pressing the Return key, mostly the OK button). A message is sent to the parent window when a click occurs on the button or the Return key is pressed.

BS_CHECKBOX ($0002) - a small rectangular button which can be marked as checked by clicking on it.

BS_AUTOCHECKBOX ($0003) - identical to BS_CHECKBOX. However, when clicked on it changes its status.

BS_RADIOBUTTON ($0004) - specifies a small round button which can be selected by clicking. Normally several radio buttons are grouped together and can be selected exclusively. Selecting one of them inactivates the remaining buttons in the same group.

BS_3STATE ($0005) - identical to BS_CHECKBOX. However, it also offers the possibility of displaying the button as gray. The graying means that button can't be selected.

BS_AUTO3STATE ($0006) - identical to BS_3STATE. However, it changes its status when clicked on.

BS_GROUPBOX ($0007) - specifies a rectangle inside which several buttons (such as radio buttons) can be grouped.

BS_AUTORADIOBUTTON $0009) - identical to BS_RADIOBUTTON. However, when activated it is automatically marked as checked and all other buttons in the same group are cleared.

BS_OWNERDRAW ($000B) - specifies a rectangle whose display is performed by a special procedure.

BS_LEFTTEXT ($0020) - displays the text left justified for check boxes and radio buttons.

In GFA-BASIC you can easily modify the appearance of buttons using the BS_OWNERDRAW style. The first character of the buttons text (given in the Button-Command or with **_Win$**()= or a system- function) determines the appearance of the button.

If it is a digit, a minus sign, or a hash sign, then the string (ignoring the hash) is taken as the decimal Handle of a bitmap. This bitmap is then used to draw the button. optionally a second bitmap handle after a comma can be given to display the button as selected. The usual windows shortcuts (as &A) can be given additionally, but are not displayed.

A leading "S" gives a softer three dimensional appearance, without the usual black border. This can be used to group buttons very close (as in the GFA-BASIC editor). The contents of the buttons title starting from the second character is diaplyed as usual.

A leading "R" displays a rounded button.

Example

```
Dialog # 1, 0, 0, 400, 300, "GFA", WS_SYSMENU
  PushButton "Command", 1, 10, 10, 140, 22,
    BS_DEFPUSHBUTTON
  PushButton "Check Box", 2, 10, 40, 140, 14,
    BS_AUTOCHECKBOX
  PushButton "Option Button 1", 3, 10, 60, 140, 14,
    BS_AUTORADIOBUTTON
  PushButton "Option Button 2", 4, 10, 75, 140, 14,
    BS_AUTORADIOBUTTON
EndDialog
ShowDialog # 1
Do : Sleep : Until Dlg_1 Is Nothing

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  If Mess% = WM_COMMAND
    Select wParam
    Case 1 : Message "Command Button Pressed" &
      #13#10 & "Option Button 2 Activated"
      SetCheck 1, 4, 1 : If Check?(1, 3) Then
        SetCheck 1, 3, 0
    Case 2 : Message "Check Box Clicked" & #13#10 &
      "Option Button 1 Activated"
      SetCheck 1, 3, 1 : If Check?(1, 4) Then
        SetCheck 1, 4, 0
    Case 3, 4 : Message "Option Button" & wParam -
      2 & " Clicked"
    EndSelect
  EndIf
EndSub

Sub Dlg_1_Close(Cancel?)
  Cancel? = False
EndSub
```

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

DefPushButton Control

Purpose

Creates a default push button control in the current active form, window, or dialog.

Syntax

DefPushbutton text$, id%, x, y, width, height[, style%]

text$:control text
id%:control identifier
x,y,b,h:iexp
style%:the control styles

Description

The control is a small rectangle with a bold outline that represents the default response for the user. The given text is displayed inside the button. The control highlights the button in the usual way when the user clicks the mouse in it and sends a message to its parent window.

style Specifies the control styles. This value can be a combination of the following styles: BS_DEFPUSHBUTTON, WS_TABSTOP, WS_GROUP, and WS_DISABLED. If you do not specify a style, the default style is BS_DEFPUSHBUTTON | WS_TABSTOP.

The command creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and

WM_NOTIFY messages should be handled in the form's _**Message** sub.

Example

```
Dlg 3D On
Local x%
Dialog # 1, 10, 10, 310, 170, "Name of the dialog"
  DefPushButton "&OK", IDOK, 10, 10, 280, 120
EndDialog
ShowDialog # 1
Do : Sleep : Until Dlg_1 Is Nothing
Dlg 3D Off

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  If Mess% = WM_COMMAND And wParam% = IDOK Then
    CloseDialog # 1
EndSub
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general Control statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

AutoCheckBox Control

Purpose

Creates a control in the current active form, window, or dialog.

Syntax

AutoCheckBox text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x,y,w,h:iexp
style%:the control styles

Description

The **AutoCheckBox** statement creates an automatic check box control. The control is a small rectangle (check box) that has the specified text displayed next to it (typically, to the right). When the user chooses the control, the control highlights the rectangle and sends a message to its parent window.

style Specifies the styles of the control. This value can be a combination of the button class style BS_AUTOCHECKBOX and the WS_TABSTOP and WS_GROUP styles. If you do not specify a style, the default style is BS_AUTOCHECKBOX | WS_TABSTOP.

Creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's _**Message** event sub.

Example

See CheckBox

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

RadioButton Control

Purpose

Creates a control in the current active form, window, or dialog.

Syntax

RadioButton text$, id%, x, y, width, height[, style%]

text$:control text
id%:control identifier
x,y,b,h:iexp
style%:the control styles

Description

The **RadioButton** statement creates an radio button control. The control is a small circle that has the given text displayed next to it, typically to its right. The control highlights the circle and sends a message to its parent window when the user selects the button. The control removes the highlight and sends a message when the button is next selected.

style Specifies styles for the automatic radio button, which can be a combination of BUTTON-class styles and the following styles: WS_TABSTOP, WS_DISABLED, and WS_GROUP. If you do not specify a style, the default style is BS_RADIOBUTTON | WS_TABSTOP.

Creates a control without an OCX wrapper; so it cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's _**Message** sub.

Example

```
Dialog # 1, 0, 0, 400, 300, "GFA", WS_SYSMENU
  RadioButton "Command", 1, 10, 10, 140, 22,
    BS_DEFPUSHBUTTON
  RadioButton "Check Box", 2, 10, 40, 140, 14,
    BS_AUTOCHECKBOX
  RadioButton "Option Button 1", 3, 10, 60, 140,
    14, BS_AUTORADIOBUTTON
  RadioButton "Option Button 2", 4, 10, 75, 140,
    14, BS_AUTORADIOBUTTON
EndDialog
ShowDialog # 1
Do : Sleep : Until Dlg_1 Is Nothing

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  If Mess% = WM_COMMAND
    Select wParam
    Case 1 : Message "Command Button Pressed" &
      #13#10 & "Option Button 2 Activated"
      SetCheck 1, 4, 1 : If Check?(1, 3) Then
        SetCheck 1, 3, 0
    Case 2 : Message "Check Box Clicked" & #13#10 &
      "Option Button 1 Activated"
      SetCheck 1, 3, 1 : If Check?(1, 4) Then
        SetCheck 1, 4, 0
    Case 3, 4 : Message "Option Button" & wParam -
      2 & " Clicked"
    EndSelect
  EndIf
EndSub

Sub Dlg_1_Close(Cancel?)
  Cancel? = False
```

```
EndSub
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

AutoRadioButton Control

Purpose

Creates a control in the current active form, window, or dialog.

Syntax

AutoRadioButton text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

The **AutoRadioButton** statement creates an automatic radio button control. This control automatically performs mutual exclusion with the other **AutoRadioButton** controls in the same group. When the button is chosen, the application is notified with BN_CLICKED.

style Specifies styles for the automatic radio button, which can be a combination of BUTTON-class styles and the following styles: WS_TABSTOP, WS_DISABLED, and WS_GROUP. If you do not specify a style, the default style is BS_AUTORADIOBUTTON | WS_TABSTOP.

Creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's _**Message** sub.

Example

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general Control statement any control type can be created.

See Also

Controll, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

EditText Control

Purpose

Creates an edit field control for user input.

Syntax

EditText text$, ID%, x%, y%, w%, h%[,style%]

Description

An **EditText** element is a rectangular area inside which text can be entered and edited (using Backspace and Delete). Clicking on an **EditText** element displays a text cursor within the rectangle. EditText can contain the WS_TABSTOP, WS_GROUP, WS_VSCROLL, WS_HSCROLL and WS_DISABLED style elements.

Other styles:

EDITES_LEFT	($0000)	sets the text left justified in the edit field
ES_CENTER	($0001)	centers the text within a multi-line edit field.
ES_RIGHT	($0002)	sets the text right justified within a multi-line edit field.
ES_MULTILINE	($0004)	defines a multi-line edit field.
ES_UPPERCASE	($0008)	converts all characters in the IBM US character set to uppercase.

ES_LOWERCASE	($0010)	converts all characters in the IBM US character set to lowercase.
ES_PASSWORD	($0020)	displays all entered characters as asterisk.
ES_AUTOVSCROLL	($0040)	scrolls the text one page up when the user presses the Return key on the last line.
ES_AUTOHSCROLL	($0080)	when further characters are entered at the end of the line, scrolls the text ten characters to the left. Pressing the Return key sets the text back to position zero.
ES_NOHIDESEL	($0100)	makes the selected entry in an edit field permanently visible.
ES_OEMCONVERT	($0400)	converts characters from ANSI into OEM and back (for example using your own character table).

Example

```
Dlg 3D On
Global style%, style2%, file$
Dlg Base Unit
style% = WS_BORDER | WS_TABSTOP
style2% = BS_DEFPUSHBUTTON | WS_TABSTOP
Dialog # 1, 10, 10, 150, 100, "Test-Dialog"
 EditText "", 101, 50, 10, 80, 14, style%
 PushButton "OK", IDOK, 10, 60, 40, 14, style2%
 PushButton "CANCEL", IDCANCEL, 80, 60, 40, 14,
   style2%
```

```
EndDialog
ShowDialog # 1
// to fill the edit field
file$ = "GFA-User"
_Win$(Dlg(1, 101)) = file$
Do
  Sleep
Until Me Is Nothing

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  Select Mess
  Case WM_COMMAND
    Select wParam
    Case IDOK
      file$ = _Win$(Dlg(1, 101))
      CloseDialog # 1
      OpenW 1
      Print file$ : Print
      Print "End with Alt + F4"
    EndSelect
  EndSelect
EndSub
```

Remarks

You can only type text into the edit field if it has the focus. The text can be read by using the **_Win$**() function and set by using **_Win$**()=.

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

ScrollBar Control

Purpose

Creates a scroll-bar control in the current active form or dialog box.

Syntax

ScrollBar text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

The **ScrollBar** statement creates a scroll-bar control. The control is a rectangle that contains a scroll box and has direction arrows at both ends. The scroll-bar control sends a notification message to its parent whenever the user clicks the mouse in the control. The parent is responsible for updating the scroll-box position. Scroll-bar controls can be positioned anywhere in a window and used whenever needed to provide scrolling input.

style% specifies a combination (or none) of the following styles: WS_TABSTOP, WS_GROUP, and WS_DISABLED. In addition to these styles, the style parameter may contain a combination (or none) of the SCROLLBAR-class styles. If you do not specify a style, the default style is SBS_HORZ.

SBS_HORZ($0000) specifies a horizontal scroll bar.

SBS_VERT($0001) specifies a vertical scroll bar.

SBS_TOPALIGN($0002) used together with SBS_HORZ, to set the scroll bar to the top of the rectangle specified in CreateWindowEx().

SBS_LEFTALIGN($0002) used together with SBS_VERT to set the scroll bars to the left side in the parent window.

SBS_BOTTOMALIGN($0004) used together with SBS_HORZ, to set the scroll bar to the bottom of the parent window.

SBS_RIGHTALIGN ($0004) used together with SBS_VERT to set the scroll bars to the right side in the parent window.

SBS_SIZEBOXTOPLEFTALIGN ($0002) used together with SBS_SIZEBOX to align the upper left corner of the Sizebox with the upper left corner of the parent window.

SBS_SIZEBOXBOTTOMRIGHTALIGN ($0004) used together with SBS_SIZEBOX to align the upper left corner of the Sizebox with the bottom right corner of the parent window.

SBS_SIZEBOX($0008) creates a Sizebox, which - as long as no SBS_SIZEBOXTOPLEFTALIGN and SBS_SIZEBOXBOTTOMRIGHTALIGN are given - has the dimensions specified in the parent window.

A scrollbar's state is set using *SetScrollRange* and *SetScrollPos* Windows API functions.

A scrollbar control doesn't post a notification to the queue, so the parent's _**Message**() event sub cannot be used to respond to scrollbar messages. Instead, the scrollbar control sends WM_HSCROLL or WM_VSCROLL messages, which are handled in the parent's _**MessageProc** event sub.

Consult the MS Windows SDK or WinApi32.Hlp for more information about the ScrollBar control.

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

AnimateCtrl Control

Purpose

Creates an **Animation** control in the current active form, window, or dialog.

Syntax

AnimateCtrl text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x,y,w,h:iexp
style%:the control styles

Description

The **AnimateCtrl** control allows you to create buttons which display animations, such as .avi files, when clicked. The control can play only AVI files that have no sound. In addition, the Animation control can display only uncompressed .avi files or .avi files that have been compressed using Run-Length Encoding (RLE).

An example of this control is the file copy progress bar in Windows 95, which uses an **AnimateCtrl** control. Pieces of paper "fly" from one folder to another while the copy operation executes.

style Specifies the styles of the control.

Creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs.

When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's _**Message** event sub.

Example

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

TabCtrl Control

Purpose

Creates a Tab control in the current active form, window, or dialog.

Syntax

TabCtrl text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x,y,w,h:iexp
style%:the control styles

Description

A *tab control* is analogous to the dividers in a notebook or the labels in a file cabinet. By using a tab control, an application can define multiple *pages* for the same area of a window or dialog box. Each page consists of a certain type of information or a group of controls that the application displays when the user selects the corresponding tab.

When the user selects a tab, a tab control sends its parent window notification messages in the form of WM_NOTIFY messages, which should be handled in the **_Message** or **_MessageProc** event sub of the parent **Form**.

Example

This example is very basic: for more information of TabStrip controls see this Windows Dev Centre page.

```
Const TCM_FIRST = &H1300
Const TCM_SETITEM = TCM_FIRST + 6
Const TCM_INSERTITEM = TCM_FIRST + 7
Const TCM_GETITEMCOUNT = TCM_FIRST + 4
Const TCIF_TEXT = 1
Const TCIF_IMAGE = 2
'
Dialog # 1, 10, 10, 400, 400, "Dialog", WS_SYSMENU
  TabCtrl "", 10, 20, 20, 150, 150
EndDialog
Dim tc1 As TCITEM, tc$ = "Tab 1"
tc1.mask = TCIF_TEXT
tc1.pszText = V:tc$
~SendMessage(DlgItem(1, 10), TCM_INSERTITEM, 1,
  tc1)
tc$ = "Tab 2"
~SendMessage(DlgItem(1, 10), TCM_INSERTITEM, 2,
  tc1)
ShowDialog # 1
'
Do
  Sleep
Until Dlg_1 Is Nothing

Sub Dlg_1_Close(Cancel?)
  Cancel? = False
EndSub

Sub frm_MessageProc(hWnd%, Mess%, wParam%,
  lParam%, retval%, ValidRet?)
  Dim hdr As Pointer NMHDR
  Switch Mess
  Case WM_NOTIFY
    Pointer(hdr) = lParam
    Print hdr.idfrom
  EndSwitch
EndSub
```

```
// Type Declarations
Type NMHDR
  hwndFrom As Long
  idfrom As Long
  code As Long
EndType
Type TCITEM
  - Long mask, dwState, dwStateMask, pszText,
    cchTextMax, iImage, lParam
EndType
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general Control statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

HeaderCtrl Control

Purpose

Creates a control in the current active form, window, or dialog.

Syntax

HeaderCtrl text$, id%, x, y, width, height[, style%]

text$:control text
id%:control identifier
x,y,b,h:iexp
style%:the control styles

Description

A header control is a window that is usually positioned above columns of text or numbers. It contains a title for each column, and it can be divided into parts. The user can drag the dividers that separate the parts to set the width of each column.

Creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's _**Message** sub.

Example

Note the requirement for the *commctrl.inc.lg32* library to run the following example.

```
'
' HeaderCtrl Example
'
$Library "..\..\Include\commctrl.inc.lg32"
Dlg 3D On
Global style%, style2%, file$
Local phdi As HD_ITEM,  hdl As HDLAYOUT, rcParent
  As RECT
Static hdrt$() : Array hdrt$() = "Column1" #10
  "Column2" #10 "Column3" #10
Dlg Base Unit
style% = WS_BORDER | HDS_BUTTONS | HDS_HORZ
style2% = BS_DEFPUSHBUTTON | WS_TABSTOP
Dialog # 1, 10, 10, 150, 100, "Test-Dialog",
  WS_SYSMENU | WS_THICKFRAME
  Dlg Base Pixel
  HeaderCtrl "", 101, 0, 0, _X, 24, style%
  phdi.Mask = HDI_FORMAT | HDI_WIDTH
  phdi.fmt = HDF_LEFT | HDF_STRING    // Left-
    justify the item
  phdi.Mask |= HDI_TEXT               // The
    .pszText member is valid
  phdi.pszText = V:hdrt$(0)           // The text
    for the item
  phdi.cxy = 75                       // The initial
    width
  phdi.cchTextMax = lstrlen(phdi.pszText)
  SendMessage Dlg(1, 101), HDM_INSERTITEM, 0,
    V:phdi
  phdi.pszText = V:hdrt$(1)           // The text
    for the 1 item
  phdi.cchTextMax = lstrlen(phdi.pszText)
  SendMessage Dlg(1, 101), HDM_INSERTITEM, 1,
    V:phdi
  DlgBase Unit
  PushButton "OK", IDOK, 10, 60, 40, 14, style2%
  PushButton "CANCEL", IDCANCEL, 80, 60, 40, 14,
    style2%
EndDialog
ShowDialog # 1
Me.AutoClose = 1
Trace Dlg_1.IsDialog
Do
  Sleep
Until Me Is Nothing

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  Dim nmhdr As Pointer NMHDR
  Select Mess
  Case WM_SIZE
    If wParam != SIZE_MINIMIZED _
      SizeW Dlg(1, 101), LoWord(lParam), 24
  Case WM_COMMAND
    Select wParam
    Case IDOK
      file$ = _Win$(Dlg(1, 101))
      CloseDialog # 1
    Case IDCANCEL
      CloseDialog # 1
    EndSelect
  Case WM_NOTIFY
    Pointer nmhdr = lParam
  EndSelect
EndSub
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

ListViewCtrl Control

Purpose

Creates a list view control in the current active form, window, or dialog.

Syntax

ListViewCtrl text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

The control is a rectangle containing a list of strings (such as filenames) from which the user can select. The **ListView** control displays items using one of four different views. You can arrange items into columns with or without column headings as well as display accompanying icons and text.

The command creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's _MessageProc sub.

Example

```
Form frm
ListViewCtrl "Listbox", 10, 20, 20, 150, 200
```

```
Do
  Sleep
Until Me Is Nothing

Sub frm_MessageProc(hWnd%, Mess%, wParam%,
  lParam%, retval%, ValidRet?)
  Dim hdr As Pointer NMHDR
  Switch Mess
  Case WM_NOTIFY
    Pointer(hdr) = lParam
    Print hdr.idfrom
  EndSwitch
EndSub
Type NMHDR
  hwndFrom As Long
  idfrom As Long
  code As Long
EndType
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

The **ListView** control is too complex to covered in full; for more information, loom at the following section of the MSDN website:

- ListView
- ListView Controls

With the general Control statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

TreeViewCtrl Control

Purpose

Creates a TreeView control in the current active form, window, or dialog.

Syntax

TreeViewCtrl text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x,y,w,h:iexp
style%:the control styles

Description

The TreeView control is designed to display data that is hierarchical in nature, such as organization trees, the entries in an index, the files and directories on a disk. Each item consists of a label and an optional bitmapped image, and each item can have a list of subitems associated with it. By clicking an item, the user can expand or collapse the associated list of subitems.

The command creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_NOTIFY message should be handled in the form's _**MessageProc** sub.

Example

```
Form frm
```

```
TreeViewCtrl "", 10, 20, 20, 150, 200
Do
  Sleep
Until Me Is Nothing

Sub frm_MessageProc(hWnd%, Mess%, wParam%,
  lParam%, retval%, ValidRet?)
  Dim hdr As Pointer NMHDR
  Switch Mess
  Case WM_NOTIFY
    Pointer(hdr) = lParam
    Print hdr.idfrom
  EndSwitch
EndSub
Type NMHDR
  hwndFrom As Long
  idfrom As Long
  code As Long
EndType
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general Control statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

ProgressCtrl Control

Purpose

Creates a Progress Bar control in the current active form, window, or dialog.

Syntax

ProgressCtrl text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

A *progress bar* is a window that an application can use to indicate the progress of a lengthy operation. It consists of a rectangle that is gradually filled with the system highlight color as an operation progresses.

A progress bar's *range* represents the entire duration of the operation, and the *current position* represents the progress that the application has made toward completing the operation.

The minimum value in the range can be from 0 to 65,535. Likewise, the maximum value can be from 0 to 65,535. If you do not set the range values, the system sets the minimum value to 0 and the maximum value to 100.

The PBM_SETPOS message sets the position to a given value. The PBM_DELTAPOS message advances the position by adding a specified value to the current position.

The PBM_SETSTEP message allows you to specify a step increment for a progress bar. Subsequently, whenever you send the PBM_STEPIT message to the progress bar, the current position advances by the specified increment. By default, the step increment is set to 10.

Example

```
Dlg 3D On // 16 bit 3D effect
Global Enum PBM_SETRANGE = WM_USER + 1, _
  PBM_SETPOS, PBM_DELTAPOS, _
  PBM_SETSTEP, PBM_STEPIT, _
  PBM_SETRANGE32, PBM_GETRANGE, _
  PBM_GETPOS, PBM_SETBARCOLOR
Local a$ = "Test", i%, j%, x%
Dialog # 1, 10, 10, 400, 200, a$
  ProgressCtrl "Hello", 10, 10, 30, _
    375, 100, WS_CHILD | WS_BORDER
EndDialog
ShowDialog # 1
SendMessage Dlg(1, 10), PBM_SETRANGE, 0,
  MakeLong(100, 0)
// PBM_SETRANGE32 only > 65536 => only for NT/2000
'SendMessage Dlg(1, 10), PBM_SETRANGE32, 0, 100
SendMessage Dlg(1, 10), PBM_SETBARCOLOR, 0,
  RGB(255, 0, 0)
DoEvents
For i% = 0 To 100
  SendMessage Dlg(1, 10), PBM_SETPOS, i%, 0
  DoEvents
  Delay 0.1 // a bit slower display
Next
Print "Please press a key"
```

```
KeyGet x%
Dlg 3D Off
CloseDialog # 1
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

TrackBarCtrl Control

Purpose

Creates a track bar control in the current active form, window, or dialog.

Syntax

TrackBarCtrl text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x,y,w,h:iexp
style%:the control styles

Description

A **TrackBarCtrl** is a window that contains a slider and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the trackbar sends notification messages to indicate the change.

A track bar notifies its parent window of user actions by sending the parent WM_HSCROLL or WM_VSCROLL messages that should be handled in the form's **_MessageProc** sub.

Example

```
Public Const TB_LINEUP                    = 0
Public Const TB_LINEDOWN                  = 1
Public Const TB_PAGEUP                    = 2
Public Const TB_PAGEDOWN                  = 3
Public Const TB_THUMBPOSITION       = 4
Public Const TB_THUMBTRACK          = 5
Public Const TB_TOP                 = 6
Public Const TB_BOTTOM              = 7
Public Const TB_ENDTRACK            = 8
Form frm
TrackBarCtrl "", 10, 20, 20, 150, 30
Global Handle hWndTrack = GetDlgItem(frm.hWnd, 10)
Do
  Sleep
Until Me Is Nothing

Sub frm_MessageProc(hWnd%, Mess%, wParam%,
  lParam%, retval%, ValidRet?)
  Switch Mess
  Case WM_HSCROLL, WM_VSCROLL
    If lParam = hWndTrack
      'Trace LoWord(wParam)
      Switch LoWord(wParam)  'Notification Message
        (Reason, sent)
      Case TB_BOTTOM           'VK_END
      Case TB_ENDTRACK         'WM_KEYUP (the user
        released a key that sent a relevant virtual
        key code)
      Case TB_LINEDOWN         'VK_RIGHT Or VK_DOWN
      Case TB_LINEUP           'VK_LEFT Or VK_UP
      Case TB_PAGEDOWN         'VK_NEXT (the user
        clicked the channel below or to the right of
        the slider)
      Case TB_PAGEUP           'VK_PRIOR (the user
        clicked the channel above or to the left of
        the slider)
      Case TB_THUMBPOSITION    'WM_LBUTTONUP following
        a TB_THUMBTRACK notification message
      Case TB_THUMBTRACK       'Slider movement (the
        user dragged the slider)
      Case TB_TOP              'VK_HOME
      EndSwitch
    EndIf
  EndSwitch
EndSub
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general Control statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

StatusCtrl Control

Purpose

Creates a Status bar control in the current active form, window, or dialog.

Syntax

StatusCtrl text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

A *status bar* is a horizontal window at the bottom of a parent window in which an application can display various kinds of status information. The status bar can be divided into parts to display more than one type of information.

If your application uses a status bar that has only one part, you can use the **_Win$**()= function to perform text operations.

Example

```
Form frm
StatusCtrl "", 10, 0, 20, _X, 30
_Win$(Dlg(frm.hWnd, 10)) = "Ready"
Do
  Sleep
```

```
Until Me Is Nothing
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general Control statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

ToolBarCtrl Control

Purpose

Creates a ToolBar control in the current active form, window, or dialog.

Syntax

ToolBarCtrl text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x,y,w,h:iexp
style%:the control styles

Description

A *toolbar* is a control window that contains one or more buttons. Each button sends a command message to the parent window when the user clicks it.

Each button has a command identifier associated with it. When the user selects a button, the toolbar sends the parent window a WM_COMMAND message that includes the command identifier of the button. The parent window examines the command identifier and carries out the command associated with the button. The WM_COMMAND message can be handled in the **_Message** or **_MessageProc** sub.

Example

```
Dialog # 1, 10, 10, 400, 200, "ToolBar",
  WS_SYSMENU
  ToolBarCtrl "", 10, 20, 20, 150, 30
EndDialog
Local tbbut As TBBUTTON, tbs$
tbbut.cbSize = 200
tbbut.pszText = Len(tbs$)
tbbut.cchText = V:tbs$
// For some reason, this fails to print a
  button...
tbs$ = "Button 1" : ~SendMessage(DlgItem(1, 10),
  TB_INSERTBUTTON, 1, V:tbbut)
ShowDialog # 1
Do
  Sleep
Until Dlg_1 Is Nothing

Sub Dlg_1_Close(Cancel?)
  Cancel? = False
EndSub
Global Const TB_INSERTBUTTON = (WM_USER + 21)
Type TBBUTTON2
  - Int iBitmap, idCommand
  - Byte fsState, fsStyle, bReserved
  - Long dwData, iString
EndType
Type TBBUTTON
  cbSize As Long
  dwMask As Long
  idCommand As Long
  iImage As Long
  fsState As Byte
  fsStyle As Byte
  cx As Word
  lParam As Long
  pszText As Long
  cchText As Long
```

```
End Type
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general Control statement any control type can be created.

For the full range of Windows messages and constants, see this page; the constant values (some of which are not declared in GB32) are listed below:

Toolbar button styles
Const TBSTYLE_BUTTON = $0000
Const TBSTYLE_SEP = $0001
Const TBSTYLE_CHECK = $0002
Const TBSTYLE_GROUP = $0004
Const TBSTYLE_CHECKGROUP = (TBSTYLE_GROUP Or TBSTYLE_CHECK)
Const TBSTYLE_DROPDOWN = $0008
Const TBSTYLE_AUTOSIZE = $0010
Const TBSTYLE_NOPREFIX = $0020
Const TBSTYLE_TOOLTIPS = $0100
Const TBSTYLE_WRAPABLE = $0200
Const TBSTYLE_ALTDRAG = $0400
Const TBSTYLE_FLAT = $0800
Const TBSTYLE_LIST = $1000
Const TBSTYLE_CUSTOMERASE = $2000
Const TBSTYLE_REGISTERDROP = $4000
Const TBSTYLE_TRANSPARENT = $8000
Const TBSTYLE_DRAWDDARROWS = $00000001

ToolBar Ex Styles
Const TBSTYLE_EX_DRAWDDARROWS = $1

Const TBSTYLE_EX_HIDECLIPPEDBUTTONS = $10
Const TBSTYLE_EX_DOUBLEBUFFER = $80

ToolBar Messages (where Const WM_USER = $0400)
Const TB_ENABLEBUTTON = (WM_USER + 1)
Const TB_CHECKBUTTON = (WM_USER + 2)
Const TB_PRESSBUTTON = (WM_USER + 3)
Const TB_HIDEBUTTON = (WM_USER + 4)
Const TB_INDETERMINATE = (WM_USER + 5)
Const TB_MARKBUTTON = (WM_USER + 6)
Const TB_ISBUTTONENABLED = (WM_USER + 9)
Const TB_ISBUTTONCHECKED = (WM_USER + 10)
Const TB_ISBUTTONPRESSED = (WM_USER + 11)
Const TB_ISBUTTONHIDDEN = (WM_USER + 12)
Const TB_ISBUTTONINDETERMINATE= (WM_USER + 13)
Const TB_ISBUTTONHIGHLIGHTED = (WM_USER + 14)
Const TB_SETSTATE = (WM_USER + 17)
Const TB_GETSTATE = (WM_USER + 18)
Const TB_ADDBITMAP = (WM_USER + 19)
Const TB_ADDBUTTONSA = (WM_USER + 20)
Const TB_INSERTBUTTONA = (WM_USER + 21)
Const TB_ADDBUTTONS = (WM_USER + 20)
Const TB_INSERTBUTTON = (WM_USER + 21)
Const TB_DELETEBUTTON = (WM_USER + 22)
Const TB_GETBUTTON = (WM_USER + 23)
Const TB_BUTTONCOUNT = (WM_USER + 24)
Const TB_COMMANDTOINDEX = (WM_USER + 25)
Const TB_SAVERESTOREA = (WM_USER + 26)
Const TB_CUSTOMIZE = (WM_USER + 27)
Const TB_ADDSTRINGA = (WM_USER + 28)
Const TB_GETITEMRECT = (WM_USER + 29)
Const TB_BUTTONSTRUCTSIZE = (WM_USER + 30)
Const TB_SETBUTTONSIZE = (WM_USER + 31)
Const TB_SETBITMAPSIZE = (WM_USER + 32)
Const TB_AUTOSIZE = (WM_USER + 33)
Const TB_GETTOOLTIPS = (WM_USER + 35)

```
Const TB_SETTOOLTIPS = (WM_USER + 36)
Const TB_SETPARENT = (WM_USER + 37)
Const TB_SETROWS = (WM_USER + 39)
Const TB_GETROWS = (WM_USER + 40)
Const TB_GETBITMAPFLAGS = (WM_USER + 41)
Const TB_SETCMDID = (WM_USER + 42)
Const TB_CHANGEBITMAP = (WM_USER + 43)
Const TB_GETBITMAP = (WM_USER + 44)
Const TB_GETBUTTONTEXTA = (WM_USER + 45)
Const TB_GETBUTTONTEXTW = (WM_USER + 75)
Const TB_REPLACEBITMAP = (WM_USER + 46)
Const TB_SETINDENT = (WM_USER + 47)
Const TB_SETIMAGELIST = (WM_USER + 48)
Const TB_GETIMAGELIST = (WM_USER + 49)
Const TB_LOADIMAGES = (WM_USER + 50)
Const TB_GETRECT = (WM_USER + 51)
Const TB_SETHOTIMAGELIST = (WM_USER + 52)
Const TB_GETHOTIMAGELIST = (WM_USER + 53)
Const TB_SETDISABLEDIMAGELIST = (WM_USER + 54)
Const TB_GETDISABLEDIMAGELIST = (WM_USER + 55)
Const TB_SETSTYLE = (WM_USER + 56)
Const TB_GETSTYLE = (WM_USER + 57)
Const TB_GETBUTTONSIZE = (WM_USER + 58)
Const TB_SETBUTTONWIDTH = (WM_USER + 59)
Const TB_SETMAXTEXTROWS = (WM_USER + 60)
Const TB_GETTEXTROWS = (WM_USER + 61)
Const TB_GETOBJECT = (WM_USER + 62)
Const TB_GETBUTTONINFOW = (WM_USER + 63)
Const TB_SETBUTTONINFOW = (WM_USER + 64)
Const TB_GETBUTTONINFOA = (WM_USER + 65)
Const TB_SETBUTTONINFOA = (WM_USER + 66)
Const TB_INSERTBUTTONW = (WM_USER + 67)
Const TB_ADDBUTTONSW = (WM_USER + 68)
Const TB_HITTEST = (WM_USER + 69)
Const TB_SETDRAWTEXTFLAGS = (WM_USER + 70)
Const TB_GETHOTITEM = (WM_USER + 71)
Const TB_SETHOTITEM = (WM_USER + 72)
Const TB_SETANCHORHIGHLIGHT = (WM_USER + 73)
Const TB_GETANCHORHIGHLIGHT = (WM_USER + 74)
Const TB_SAVERESTOREW = (WM_USER + 76)
Const TB_ADDSTRINGW = (WM_USER + 77)
Const TB_MAPACCELERATORA = (WM_USER + 78)
Const TB_GETINSERTMARK = (WM_USER + 79)
Const TB_SETINSERTMARK = (WM_USER + 80)
Const TB_INSERTMARKHITTEST = (WM_USER + 81)
Const TB_MOVEBUTTON = (WM_USER + 82)
Const TB_GETMAXSIZE = (WM_USER + 83)
Const TB_SETEXTENDEDSTYLE = (WM_USER + 84)
Const TB_GETEXTENDEDSTYLE = (WM_USER + 85)
Const TB_GETPADDING = (WM_USER + 86)
Const TB_SETPADDING = (WM_USER + 87)
Const TB_SETINSERTMARKCOLOR = (WM_USER + 88)
Const TB_GETINSERTMARKCOLOR = (WM_USER + 89)
```

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

UpDownCtrl Control

Purpose

Creates an UpDown common control in the current active form, window, or dialog.

Syntax

UpDownCtrl text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x,y,w,h:iexp
style%:the control styles

Description

An **UpDown** control has a pair of arrow buttons which the user can click to increment or decrement a value, such as a scroll position or a value in an associated control, known as a buddy control.

The command creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's _MessageProc sub.

Example

```
/* Styles for the UpDown Control
Global Enum UDS_WRAP = 1, _
 UDS_SETBUDDYINT, UDS_ALIGNRIGHT=4, _
```

```
  UDS_ALIGNLEFT=8, UDS_AUTOBUDDY=10, _
  UDS_ARROWKEYS=$20, UDS_HORZ =$40, _
  UDS_NOTHOUSANDS=$80, UDS_HOTTRACK =$100
/* Messages to Control the animation
Global Enum UDM_SETRANGE=WM_USER + 101, _
  UDM_GETRANGE, UDM_SETPOS, UDM_GETPOS, _
  UDM_SETBUDDY, UDM_GETBUDDY, UDM_SETACCEL, _
  UDM_GETACCEL, UDM_SETBASE, UDM_GETBASE, _
  UDM_SETRANGE32, UDM_GETRANGE32, _
  UDM_SETUNICODEFORMAT=$2005, _
  UDM_GETUNICODEFORMAT=$2006
OpenW 1
Ocx TextBox ed1 = "", 10, 10, 100, 20
ed1.Appearance = 1
UpDownCtrl"", 1010, 10, 10, 100, 20, _
  UDS_ARROWKEYS | UDS_WRAP | UDS_SETBUDDYINT |
    UDS_ALIGNLEFT | WS_TABSTOP
Local hUpDown As Handle =  Dlg(Win_1.hWnd, 1010)
SendMessage hUpDown, UDM_SETBUDDY, ed1.hWnd, 0
SendMessage hUpDown, UDM_SETRANGE, 0,
  MakeLong(1000, 990)
SendMessage hUpDown, UDM_SETPOS, 0, MakeLong(0,
  993)
~SetFocus(Dlg(Win_1.hWnd, 10))
Do
  Sleep
Until Me Is Nothing

Sub Win_1_MessageProc(hWnd%, Mess%, wParam%,
  lParam%, retval%, ValidRet?)
  Dim hdr As Pointer NMHDR
  Switch Mess
  Case WM_NOTIFY
    Pointer(hdr) = lParam
  EndSwitch
EndSub
Type NMHDR
```

```
  hwndFrom As Long
  idfrom As Long
  code As Long
EndType
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general Control statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

RichEditCtrl Control

Purpose

Creates a rich edit control.

Syntax

RichEditCtrl text$, ID%, x%, y%, w%, h%[,style%]

Description

A “rich edit control” is a window in which the user can enter and edit text. The text can be assigned character and paragraph formatting, and can include embedded OLE objects. Rich edit controls provide a programming interface for formatting text. However, an application must implement any user interface components necessary to make formatting operations available to the user.

Rich edit controls support almost all of the operations and notification messages used with multiline edit controls. Thus, applications that already use edit controls can be easily changed to use rich edit controls. Additional messages and notifications enable applications to access the functionality unique to rich edit controls. For information about edit controls, see EditText control.

style%:

ES_LEFT - sets the text left justified in the edit field

ES_CENTER - centers the text within a multi-line edit field.

ES_RIGHT - sets the text right justified within a multi-line edit field.

ES_MULTILINE - defines a multi-line edit field.

ES_AUTOVSCROLL - scrolls the text one page up when the user presses the Return key on the last line.

ES_AUTOHSCROLL - when further characters are entered at the end of the line, scrolls the text ten characters to the left. Pressing the Return key sets the text back to position zero.

ES_NOHIDESEL - makes the selected entry in an edit field permanently visible.

ES_DISABLENOSCROLL - Disables scrollbars instead of hiding them when they are not needed.

ES_EX_NOCALLOLEINIT - Prevents the control from calling theOleInitialize function when created. Useful only in dialog templates because CreateWindowEx does not accept this style.

ES_NOIME - Disables the input method editor (IME) operation. Available for Asian-languages only.

ES_SAVESEL - Preserves the selection when the control loses the focus. By default, the entire contents of the control are selected when it regains the focus.

ES_SELFIME - Directs the rich edit control to allow the application to handle all IME operations. Available for Asian-languages only.

ES_SUNKEN - Displays the control with a sunken border style so that the rich edit control appears recessed into its

parent window.
Windows 95: Applications developed for Windows 95 should use WS_EX_CLIENTEDGE instead of ES_SUNKEN.

ES_VERTICAL - Draws text and objects in a vertical direction. Available for Asian-languages only.

Rich edit controls support most of the notification messages used with edit controls, plus some more. Use the WinApi32.Hlp or MS Windows SDK to get more information about Rich edit controls.

Example

```
Dlg 3D On
Global style%, style2%, file$
Dlg Base Unit
style% = WS_BORDER | WS_TABSTOP
style2% = BS_DEFPUSHBUTTON | WS_TABSTOP
Dialog # 1, 10, 10, 150, 100, "Test-Dialog"
  RichEditCtrl "", 101, 50, 10, 80, 14, style%
  PushButton "OK", IDOK, 10, 60, 40, 14, style2%
  PushButton "CANCEL", IDCANCEL, 80, 60, 40, 14,
    style2%
EndDialog
ShowDialog # 1
// to fill the edit field
file$ = "GFA-User"
_Win$(Dlg(1, 101)) = file$
Do
  Sleep
Until Me Is Nothing

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  Select Mess
  Case WM_COMMAND
    Select wParam
    Case IDOK
      file$ = _Win$(Dlg(1, 101))
      CloseDialog # 1
      OpenW 1
      Print file$ : Print
      Print "End with Alt + F4"
    EndSelect
  EndSelect
EndSub
```

Remarks

You can only type text into the edit field if it has the focus. The text can be read by using the **_Win$**() function and set by using **_Win$**()=.

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Static Command

Purpose

Defines local variables in a subroutine and main program. Variables declared with the **Static** statement retain their values as long as the code is running.

Syntax

Static [**Dim**] varname[()] [**As** [**New**] type] [= value], …

Static type varname1 [= value], varname2 [= value], …

Static varname1**$** [= value], varname2**%** [= value], …

varname: name of variable

*type: Optional. Data type of the variable; may be **Byte**, **Boolean**, **Card**, **Short**, **Word**, **Integer**, **Long**, **Large**, **Currency**, **Single**, **Double**, **Date**, **String**, (for variable-length strings), **String** * length (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. Use a separate **As** type clause for each variable being defined.*

Description

Static declares local variables. When used in the main program, the variable's scope is limited to the main part and isn't known in subroutines. In this respect, **Static** and **Local** work the same.

The **New** keyword enables implicit creation of a few GFA-BASIC 32 objects, like **DisAsm**, **Collection**, **StdFont, Font**, **StdPicture, Picture, CommDlg,** and **ImageList**. If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it, so you don't have to use the **Set** statement to assign the object reference. The **New** keyword can't be used to declare variables of any intrinsic data type.

If you don't specify a data type or object type and there is no **Def***type* statement in the module, the variable is **Variant** by default.

Variables can be initialized while they are declared.

When a variable isn't explicitly initialized, a numeric variable is initialized to 0, a variable-length string is initialized to a zero-length string (""), and a fixed-length string is filled with zeros. **Variant** variables are initialized to Empty. Each element of a user-defined type variable is initialized as if it were a separate variable.

Example

```
OpenW 1
AutoRedraw = 1
Local a%, x%, i%' scope in main program
For i% = 1 To 10
  a% += i%
  Print KeepTotal(a%)
Next i%

Function KeepTotal(Number As Double)
  ' Only the variable Accumulate preserves its
    value between calls.
  Static Accumulate As Double
  Accumulate = Accumulate + Number
  KeepTotal = Accumulate
End Function
```

See Also

Global, Dim, Local

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

LoadForm Command, Load Event

Action:

LoadForm loads a **Form** designed in the Form Editor, which initiates a **Load** event.

Syntax

LoadForm frm [options] [, x, y]

frm:Form object
options:[**Center** | **Client3D** | **Full** | **Default** | **Hidden** | **Tool** | **Help** | **Top** | **Palette** | **Fixed** | **NoCaption** | **NoTitle**]
x, y:iexp

Sub *Form*_**Load** [(Index%)]

Description

LoadForm *name* loads a Form which was designed earlier in the GFA-BASIC 32 Form editor. The *name* must be the name given in properties window. Eventually, the **Load** event sub is invoked.

At design time the initial layout of the form can be determined using the Form's **StartUpMode** and **Visible** properties. However not all the attributes of a window can be set at design time. To overcome this limitation a number of flags can be specified in the **LoadForm** command. These flags allow you to initially center the window or create full screen window.

At design time you can set the **Owned** property determining that the form is to be loaded as an owned window. When set and when executing **LoadForm**, the form will be owned by the current active window (**Me**). When **Me** = Nothing at the time of execution of **LoadForm** the **Owned** property is ignored.

The **Owned** property permits you to specify that the form being shown is to be owned by the current active form. When you use this option, you achieve two interesting effects: the owned form is always shown in front of its owner (parent), even if the parent has the focus, and when the parent form is closed or minimized, all forms it owns are also automatically closed or minimized. You can take advantage of this feature to create floating forms that host a toolbar, a palette of tools, a group of icons, and so on. This technique is most effective if combine it with the window state options **Fixed** and/or **Tool**/**Palette**.

Options	**Meaning**
Center	centers the form, overrules **StartUpMode** property
Full	creates a maximized window, overrules **StartUpMode**, excludes Hidden (full windows are always visible).
Default	default, overrules **StartUpMode**
Hidden	invisible, overrules **Visible** property
Client3D	set WS_EX_CLIENTEDGE, overrules **Appearance**
Tool	creates a WS_EX_TOOLWINDOW
Help	includes a Help button in the window caption
Top	creates a top window
Palette	creates a WS_EX_PALETTEWINDOW
Fixed	a non-sizable window

NoCaption	no title bar
NoTitle	no title bar, alias

Using any of the additional parameters ignore the design time property **Visible**.

When the optional *x* and *y* are specified, the design time properties **Left** and **Top** are ignored.

The **LoadForm** command generates a **Load** event, which is not invoked immediately! The event sub is called when the form is made visible, which is not before a **DoEvents** or **Sleep** handles the events. The **Load** event sub can be used to perform initialization tasks like creating a menu, toolbar, and statusbar.

To load a **MdiChild** form, you must make sure to activate its owner/parent, the window/form with its property **MdiParent** set to True. Since **LoadForm** sets **Me,** and child windows are loaded after the parent window is created, this would hardly cause any problem.

Example

```
// To run this example you must first create a
  Form...
// ... using the Form Editor and name that form
  frm1
LoadForm frm1
Do
  Sleep
Loop Until Me Is Nothing

Sub frm1_Load
  ' Initialization code
  Global Dim mnu$()
  Array mnu$() = "&File"#10 "&New"#10 "&Open"#10
    "&Save"#10 _
    "Save &As"#10 "-"#10 "E&xit"#10 #10 _
    "&Edit"#10 "&Undo"#10 "-"#10 "Copy"#10 "Cut"#10
      "Paste"#10 #10 _
    "&Help"#10 "&About"#10 #10
  Menu mnu$()
EndSub
```

Remarks

To create a form in code use the **Form** statement or the GFA-BASIC 16 commands **OpenW**, **ChildW**, **ParentW**, and **Dialog**, they create forms as well. However, these commands do **not** generate a **Load** event, though.

See Also

Form Object, Form, OpenW, ChildW, ParentW, Dialog

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Menu() Array

Purpose

Array containing window events. Implemented for compatibility with GFA-BASIC 16. Should not be used in GFA-BASIC 32 with OCX controls.

Syntax

Menu(index)

index:iexp

Description

The **Menu**() array contains window messages from the message queue when used with **GetEvent, PeekEvent**, or **DoEvents**. (GFA-BASIC 16 compatible). The GFA-BASIC 32 COM/OCX **Sleep** command doesn't copy the messages to the **Menu**() array. **Sleep** dispatches the messages according COM rules.

About the **Menu**() array.

The menu bar created with the **Menu** *m$()* command contains the pop-up menus with the various menu entries. Such pop-up menus can be invoked in GFA-BASIC even outside of the menu bar by using the **Popup** command. The **Menu**() function receives the pop-up menus and windows created with GFA-BASIC commands like **OpenW**. **Menu**(m) returns a value indicating which event has occurred. The values are assigned as follows:

Menu(1)=1	Keyboard: NOT IMPLEMENTED in GFA-BASIC 32 Menu(5) - Used to get information about the pressed key - use **Screen_Preview** instead.
Menu(1)=4	The close box of a window was activated
Menu(1)=5	The minimum size field in a window was activated Menu(7) - New width Menu(8) - New height Menu(9) - SIZEICONC
Menu(1)=6	The maximum size field in a window was activated. Menu(7) - New width Menu(8) - New height Menu(9) - SIZEFULLSCREEN
Menu(1)=7	The arrow up box in a window was activated
Menu(1)=8	The arrow down box in a window was activated
Menu(1)=9	The arrow left box in a window was activated
Menu(1)=10	The arrow right box in a window was activated
Menu(1)=11	The area above the vertical scroll bar was activated; Page up
Menu(1)=12	The area below the vertical scroll bar was activated; Page down
Menu(1)=13	The area to the left of the horizontal scroll bar was activated; Page left
Menu(1)=14	The area to the right of the scroll bar was activated; Page right
Menu(1)=15	The vertical scroll bar was moved Menu(7) - Position in the range from 0 to

	1000
Menu(1)=16	The horizontal scroll bar was moved Menu(7) - Position in the range from 0 to 1000
Menu(1)=17	he title bar in a window was activated. If the window was moved, Menu(7) - Returns the new x-position. Menu(8) - Returns the new y-position of the upper left corner of the window.
Menu(1)=18	The size box of a window was activated. If the size of the window was changed, Menu(7) - Returns the new width. Menu(8) - Returns the new height of the window. Menu(9) - TYPEofSIZE
Menu(1)=20	A menu or a pop-up entry was selected. Menu(0) returns the index of the menu entry in the entry field or the number of the entry in a pop-up menu.
Menu(1)=21	WM_PAINT. A rectangular segment of a window must be redrawn; Redraw Message Menu(7) - Returns the left x-coordinate of the window rectangle Menu(8) - Returns the upper y-coordinate of the window rectangle Menu(9) - Returns the width of the window rectangle Menu(10) - Returns the height of the window rectangle
Menu(1)=30	Control message. A message from a Control element was sent. Menu(5) - Number of the Dialog window Menu(6) - Number of the item (ItemID) Menu(13) - The high word of lParam of the

message, e.g. LB_SELECTSTRING for a Select box, or BN_CLICK for a button.

The following always applies:

Menu(0)	The index number of the menu item selected in the current active window.
Menu(2)	Mouse x-position (corresponds to the MOUSESX function)
Menu(3)	Mouse y-position (corresponds to the MOUSESY function)
Menu(4)	The status of the mouse keys: Menu(4)=0 - No mouse key was pressed Menu(4)=1 - The left mouse button was pressed Menu(4)=2 - The right mouse button was pressed
Menu(7)	Returns the number of the GFA-BASIC window above which the mouse was located when the mouse button was pressed.
Menu(11)	Mess (message number). Same as **_Mess.**
Menu(12)	wParam. Same as **_wParam**
Menu(13)	lParam. Same as **_lParam**
Menu(14)	GFA-BASIC window handle(0-31). Same as **_winId**.
Menu(15)	Windows window handle. Same as **_hWnd**.
Menu(16)	Time in ms since booting

Example

```
Global a$, ch%, i%
Data Title &1, Entry &1, Entry &2, &End,
Data Title &2, Entry &1, Entry &2, ...,
Data Title &3, Entry &1, Entry &2, ..., , */
```

```
Dim m$(20)
i% = -1
Do
  i% ++
  Read m$(i%)
Until InStr(m$(i%), "*/")
OpenW # 1, , , , , -1
Menu m$()
Do
  DoEvents
  EvalMenu() /* MENU(1) = 20
  EvalKey() /* MENU(1) = 1
  EvalMess() /* MENU(1) = Rest
Loop

Procedure EvalMenu()
  Local e% = MENU(0)
  Local t$ = Trim$(m$(e%))
  Local p% = InStr(t$, "&")
  If e% = 3
    CloseW # 1
    End
  EndIf
  t$ = Left$(t$, p% - 1) + Mid$(t$, p% + 1)
  Cls
  Text 0, _Y / 2, t$ + " was selected"
EndProc

Procedure EvalKey
  // Does not work in GFABasic32
  Local e%, ee%
  e% = Byte(MENU(5))
  ee% = Byte(Shr(MENU(5), 8))
  WindGet 14, ch%
  Cls
  Text 0, _Y / 2, "Keyboard input"
  Text 0, _Y / 2 + ch%, "ASCII-CODE : " + Str$(e%)
  Text 0, _Y / 2 + 2 * ch%, "Scan-CODE : " +
    Str$(ee%)
EndProc 'Return

Procedure EvalMess()
  Local e%
  e% = MENU(1)
  If e%
    Cls
    Switch e%
    Case 4 : CloseW # 1 : End
    Case 5 /* Minimizer
    Case 6 /* Maximizer
    Case 7, 8, 11, 12, 15
      a$ = "vert. slider "
    Case 9, 10, 13, 14, 16
      a$ = "horz. slider "
    Case 17
      a$ = "Title line "
    Case 18
      a$ = "Sizer "
    Case 21
      a$ = "WM_PAINT message "
    EndSwitch
    If !e% = 21
      Text 0, _Y / 2, a$ + "activated"
    Else
      WindGet 14, ch%
      Text 0, _Y / 2 + ch%, a$
    EndIf
  EndIf
Return
```

See Also

MenuItem, Menu, GetEvent, PeekEvent, DoEvents

{Created by Sjouke Hamstra; Last updated: 17/10/2014 by James Gaite}

Dlg Function

Purpose

returns the window handle of a Dialog box.

Syntax

h = **Dlg**(DialogID)

h = **Dlg**(DialogID,ItemID)

DialogID, ItemID: iexp
h: Handle

Description

Dlg(*DialogID*) returns the window handle of a previously opened Dialog box. The parameter is the number used in the **Dialog #** command.

If *DialogID* contains a number of a Dialog window previously opened with **Dialog #** and *ItemID* is the number of a Dialog item **DlgItem**(*DialogID*,*ItemID*) returns the Windows handle of the item.

Example

```
Dialog # 1, 10, 10, 100, 150, "This is a Dialog",
  128
  PushButton "Ok", IDOK, 10, 10, 80, 20
  PushButton "Cancel", IDCANCEL, 10, 30, 80, 20
  LText "Cancel", 21, 10, 50, 80, 40
EndDialog
```

```
_Win$(Dlg(1, 21)) = "Label"
_Win$(Dlg(1)) = "Dialog box title"
ShowDialog # 1
```

Remarks

DlgItem(*DialogID*,*ItemID*) is a synonym for **Dlg**(*DialogID*,*ItemID*) and can be used instead.

See Also

Dialog, DlgItem()

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

DlgItem Function

Purpose

returns the handle of a Dialog item.

Syntax

h = **DlgItem**(DialogID,ItemID)

DialogID, ItemID:aexp
h:iexp

Description

If *DialogID* contains a number of a Dialog window previously opened with **Dialog** and *ItemID* is the number of a Dialog item **DlgItem**(*DialogID*,*ItemID*) returns the Windows handle of the item.

Example

```
Dialog # 1, 10, 10, 100, 150, "This is a Dialog",
  128
  PushButton "Ok", IDOK, 10, 10, 80, 20
  PushButton "", IDCANCEL, 10, 30, 80, 20
  LText "Cancel", 21, 10, 50, 80, 40
EndDialog
_Win$( DlgItem(1, IDCANCEL) ) = "Cancel"
_Win$( DlgItem(1, 21) ) = "Label"
ShowDialog # 1
```

Remarks

Dlg(*DialogID*,*ItemID*) is a shortcut for **DlgItem**(*DialogID*,*ItemID*) and can be used instead.

See Also

Dlg(), Dialog

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

_Win$ Function

Purpose

Returns or changes the text of a window or control

Syntax

_**Win$**(hWnd)=x$

x$ = _**Win$**(hWnd)

hWnd:window handle
x$:svar

Description

Corresponds somewhat to **TitleW**, but instead of a window number you must specify a window handle (e.g. **Win**(1) or **Win_1.hWnd**). In contrast to **TitleW**, you can also change the title/contents of Controls (e.g. EditText).

Example

```
OpenW # 1
Dim x$ = _Win$(Win_1.hWnd)
If x$ != "Win #1"
  _Win$(Win(1)) = "Win #1"
  Print _Win$(Win(1))
EndIf
Do
  Sleep
Loop Until Me Is Nothing
```

Remarks

The OCX forms and controls have a property to set the text or caption. _**Win$**() is a shortcut for the APIs *SetWindowText* and *GetWindowText.*

See Also

TitleW, Caption

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

MoveW Command

Purpose

Moves a window.

Syntax

MoveW #n, x, y

Description

MoveW #n, x, y moves the window specified in n (0 to **_maxInt**) so that its upper left corner is at coordinates x, y in pixels. When the window doesn't have a number, the handle can be specified.

Example

```
Local a%
OpenW # 200, 15, 15, 200, 100, -1
Print "Press any key to move this window"
KeyGet a%
MoveW 200, 50, 50   ' pixels
Print AT(1, 1); "Press any key to close this
  window"
KeyGet a%
CloseW # 200
```

Draws two windows on the screen and waits for a key press. The second window is then moved.

Remarks

Any window or control can be moved. For Ocx objects you can also use the **Move** method, but for forms this method takes the coordinates in twips.

See Also

OpenW, SizeW, Move

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

SizeW Command

Purpose

Changes the size of a window.

Syntax

SizeW [#]n, w, h

n, w, h:integer expression

Description

SizeW #n, w, h changes the size of the window specified in n (0 to _maxInt). w specifies the new width and h the new height in pixels. The position of the window (upper left corner) remains unchanged. When the window doesn't have a number, the handle can be specified.

Example

```
Local a%
OpenW # 1, 10, 10, 240, 100, -1
Print "Press any key to increase the window size"
KeyGet a%
SizeW # 1, 400, 200
```

Draws a window on the screen and waits for a key press. The window size is then changed.

Remarks

Any window or control can be sized. For Ocx objects you can also use the **Move** method, but for forms this method takes the coordinates in twips.

See Also

OpenW, CloseW, MoveW

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

CloseW Command

Purpose

Closes a window.

Syntax

CloseW [#]wh

wh:integer expression

Description

Closes a window with window number wh (0 to _maxInt). If the window doesn't have a number, a window handle can be passed.

Example

```
Local a%
OpenW # 100, 10, 10, 200, 100, -1
KeyGet a%
CloseW 100
Debug.Show

Sub Form_Destroy(Index%)
  Debug "Destroy event"
EndSub
```

Draws a window on the screen. When a key is pressed, the window is closed again.

Remarks

Unlike the **Close** method used with forms, if the window being closed with **CloseW** has already been closed, does not exist or has in some other way been set to Nothing, the command is ignored - no error is raised - and the program operation is not interrupted.

See Also

FullW, ClearW, OpenW, TitleW, SizeW, TopW

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

ClearW Command

Purpose

Deletes the contents of a window.

Syntax

ClearW [#]n

Description

ClearW n deletes the contents of the window with number n (0 to _maxInt). If the window doesn't have a number, the handle can be passed.

Example

```
OpenW 1
Local a%, i%, x%
For i% = 1 To 500
  Color Rand(_C)
  Line Rand(300), Rand(300), Rand(300), Rand(300)
  Circle Rand(300), Rand(300), Rand(300)
Next
Color 0 : FontSize = 40
Text 50, _Y - 150, "Please press a key"
KeyGet a%
ClearW 1
FontSize = 20
Text _X / 2 - _X / 3, _Y / 2 - 20, "Window
  contents will be deleted"
KeyGet a%
CloseW 1
```

Draws a window with lines and circles. After pressing a key the window is cleared and the text "Window contents will be deleted..." is written in the window. When a key is pressed again the window is closed.

Remarks

ClearW doesn't work in the event sub **Form_Paint**. It generates a WM_PAINT causing an endless loop.

See Also

FullW, CloseW, OpenW, TitleW, SizeW, TopW

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

ShowW Command

Purpose

Displays a window with a certain status.

Syntax

ShowW w, stat%

w%, stat%:integer expression

Description

The **ShowW** command is used to display a window with a particular status. This includes for example not displaying the window at all - i.e. show it as an icon or "invisible". The command requires two parameters: The first (w) specifies the window number or window handle (frm1.hWnd), the second (stat%) gives the status for the window to be displayed with. stat% must take one of the following values:

SW_HIDE (0) - hides a window and redirects its input to the next one.

SW_MINIMIZE (6) - minimizes (iconizes) a window and activates the Top Level window from the window manager list.

SW_RESTORE (9) - same as SW_SHOWNORMAL

SW_SHOW (5) - activates a window and places it at the current position using the current dimensions.

SW_SHOWMAXIMIZED (3) - activates a window and uses its maximum dimensions.

SW_SHOWMINIMIZED (2) - activates the window and displays it as an icon.

SW_SHOWMINNOACTIVE (7) - displays a window as an icon but keeps the current window active.

SW_SHOWNA (8) - displays a window using its current status but keeps the current window active.

SW_SHOWNOACTIVATE (4) - displays a window at its last position and in latest dimension but keeps the current window active.

SW_SHOWNORMAL (1) - activates a window and displays it. For maximized and minimized windows Windows restores the previous position and dimension.

Example

```
OpenW # 1
OpenW # 2
ShowW 1, SW_SHOWMINIMIZED
Do
  GetEvent
Until MouseK %& 2
CloseW # 2
CloseW # 1
```

Opens window 1 and minimizes it...

Remarks

The command **ShowW** corresponds to Windows function *ShowWindow*().

See Also

Iconic?(), Visible?(), Zoomed?()

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

EnableW Command

Purpose

Enables mouse and keyboard input for a window.

Syntax

EnableW wh%

wh%:integer expression

Description

The mouse and keyboard input for windows can be controlled separately. **EnableW** enables these inputs for the window specified in wh%, **DisableW** disables them.

wh% is a value between 0 and 31 to identify a window, or a window handle.

Example

```
OpenW 1
Ocx Command cmd = "Click Me", 100, 50, 100, 22
DisableW 1
Print "Window Disabled - Status = "; Enabled?(1)
Local t As Double = Timer
While Timer - t < 5
  Print AT(1, 2); "Window disabled for "; Int(5.99
    - (Timer - t)); " seconds"
Wend
EnableW 1
Print "Window Re-enabled - Status = "; Enabled?(1)
```

```
Do : Sleep : Until Me Is Nothing

Sub cmd_Click
  Message "Button Clicked"
EndSub
```

Now input (mouse & keyboard) is possible again. The input for window 1 is first deactivated and the reactivated.

See Also

DisableW, Enabled?()

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

DisableW Command

Purpose

disables mouse and keyboard input for a window.

Syntax

DisableW wh%

wh%:integer expression

Description

Disables mouse and keyboard input for the window specified in wh%, **EnableW** enables them.

wh% can be a window number (**OpenW**, **ChildW**, **ParentW**) or an API window handle.

Example

```
OpenW 1
Ocx Command cmd = "Click Me", 100, 50, 100, 22
DisableW 1
Print "Window Disabled - Status = "; Enabled?(1)
Local t As Double = Timer
While Timer - t < 5
  Print AT(1, 2); "Window disabled for "; Int(5.99
    - (Timer - t)); " seconds"
Wend
EnableW 1
Print "Window Re-enabled - Status = "; Enabled?(1)
Do : Sleep : Until Me Is Nothing
```

```
Sub cmd_Click
 Message "Button Clicked"
EndSub
```

See Also

EnableW, Enabled?()

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

Enabled? Function

Purpose

ReturnsTrue if the window or Ocx object is enabled.

Syntax

Enabled?(wh%)

wh%:integer expression

Description

The single parameter (wh%) in this function specifies the number of the window whose status is to be returned. When 0 <= wh% <= 31, wh% specifies a window number, otherwise is holds a window or Ocx object handle.

Example

```
OpenW 1, 0, 0, 200, 200 : Win_1.Caption = "Win_1"
Ocx Command cmd1 = "Enabled", 10, 10, 100, 22
OpenW 2, 250, 0, 200, 200 : Win_2.Caption =
 "Win_2"
Print "Disabled" : Win_2.Enabled = False
Debug.Show
~SetWindowPos(Debug.hWnd, 0, 500, 0, 300, 200, 0)
Trace Enabled?(cmd1.hWnd)   // Prints True
Trace Enabled?(2)           // Prints False
Do : Sleep : Until Win_1 Is Nothing
CloseW 2
Debug.Hide
```

Remarks

Enabled? corresponds to Windows function *IsEnabled*().

See Also

Iconic?(), Visible?(), Zoomed?(), WindowState

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

SetCheck Command

Purpose

Marks a checkbox or radio button

Syntax

SetCheck dlg, item, flag

dlg, item, flag:integer expression

Description

SetCheck is the opposite of **Check?**(). With **SetCheck** 1,40,1 the checkbox or radio button in the Dialog box #1 with ID=40 is marked 1.

Example

```
Dialog # 1, 10, 10, 200, 100, "Testdialog",
  WS_SYSMENU
  AutoCheckBox "Checkbox", 100, 10, 10, 100, 20
  LText "Checked", 101, 10, 40, 70, 16
EndDialog
ShowDialog # 1
// mark the CheckBox with the ID 100
// in the Dialog # 1
SetCheck 1, 100, 1
Do
  Sleep
Until Dlg_1 Is Nothing

Sub Dlg_1_Close(Cancel?)
```

```
  Cancel? = False
EndSub

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  If Mess% = WM_COMMAND And wParam% = 100
    _Win$(Dlg(1, 101)) = (Check?(1, 100) ?
     "Checked" : "Unchecked")
  EndIf
EndSub
```

See Also

Checkbox, RadioButton, Check?

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Check? Function

Purpose

Interrogates a (**Auto**)**CheckBox**, a (**Auto**)**RadioButton**, or a BS_3STATEBOX button.

Syntax

a%=**Check?**(Dlg, item)

Dlg: ivar
item: ivar

Description

When a 0 is returned, it is not marked. When a 1 is returned, it is marked with a check mark or a cross. In case of a 3STATEBOX , a 2 is returned when it is filled.

Dlg is the number of the Dialog box or a window handle.
item is the number (ID) of the button/checkbox.

Example

```
Dialog # 1, 10, 10, 200, 100, "Testdialog"
  AutoCheckBox "Checkbox", 100, 10, 10, 100, 20
EndDialog
ShowDialog # 1
SetCheck 1, 100, 1 ' check Checkbox ID = 100 in
  Dialog #1
Do
  Sleep
Until Check?(1, 100) = 0
```

```
Message "Selected"
CloseDialog # 1
```

Remarks

This command is most useful in a GFA Editor Extension when creating a user interface with the Dialog command.

See Also

SetCheck

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Zoomed? Function

Purpose

Returns True if the window is maximized.

Syntax

Zoomed?(wh%)

wh%:integer expression

Description

The single parameter (wh%) in this function specifies the number of the window whose status is to be returned. When 0 <= wh% <= 31, wh% specifies a window number, otherwise is holds a window handle.

Example

```
OpenW # 1 : AutoRedraw = 1
ShowW 1, SW_SHOWMAXIMIZED
OpenW # 2, 200, 200, 400, 200
Win 2
Print Zoomed?(2)// False
Win 1
Print Zoomed?(1)// True
```

Remarks

Zoomed? corresponds to Windows function *IsZoomed*().

See Also

Enabled?(), Iconic?(), Visible?(), WindowState

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Visible? Function

Purpose

Returns True if the window/Form is visible.

Syntax

? = **Visible?**(wh%)

wh%:integer expression

Description

The only parameter *wh%* for this function specifies the number or the handle of the window for which the status is to be returned.

Example

```
OpenW 1, 10, 10, 300, 200 : TitleW 1, "Window 1"
Ocx CheckBox chk = "Show Window 2", 10, 10, 120,
  14
Ocx Label lbl = "Window 2 is Invisible", 10, 30,
  120, 14
OpenW Hidden 2, 320, 10, 300, 200 : TitleW 2,
  "Window 2"
Do : Sleep : Until Win_1 Is Nothing Or Win_2 Is
  Nothing
CloseW 1 : CloseW 2

Sub chk_Click
  Win_2.Visible = -chk.Value
  lbl.Caption = "Window 2 is " & (Visible?
    (Win_2.hWnd) ? "Visible" : "Invisible")
EndSub
```

Remarks

For forms it is easier to inspect the **Visible** property.

Visible?() corresponds to the Windows function *IsWindowVisible*()

See Also

Enabled?(), Iconic?(), Zoomed?(), WindowState

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Iconic? Function

Purpose

Returns True if the window is iconized.

Syntax

Bool = **Iconic**?(wh%)

wh%:integer expression

Description

The single parameter (wh%) in this function specifies the number of the window whose status is to be reported.

Example

```
Debug.Show
OpenW # 1
ShowW 1, SW_SHOWMINIMIZED
OpenW # 2
Trace Iconic?(1)                    // True
Trace Iconic?(2)                    // False
Trace IsIconic(Win_1.hWnd)          // 1 (True)
Trace IsIconic(Win_2.hWnd)          // 0 (False)
Trace Me.WindowState                // basNormal (0)
CloseW 2
CloseW 1
```

Remarks

Iconic?() corresponds to Windows function *IsIconic*() which is implemented as an API and takes the windows handle as in the example above.

See Also

Enabled?(), Visible?(), Zoomed?(), WindowState

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

True Variable

Purpose

Constant keyword for logical true = -1.

Syntax

True

Description

Contains the value for logical true.

Example

```
Auto a!, i%
OpenW # 1
i% = 20
If i%
  a! = True
  Print "i% is not equal to 0; a!="; a!
EndIf
i% = 0
If !i%
  a! = False
  Print "i% is equal to 0; a!="; a!
EndIf
```

Prints:

```
i% is not equal to 0; a!=-1
i% is equal to 0; a!=0
```

See Also

False

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Try Command

Purpose

Local error handling

Syntax

Try

// code

Catch

// error handler

EndCatch

Description

Try and **Catch**/**EndCatch** appears always as a pair inside a procedure or function. The statements after **Try** are executed as usual, and the part between **Catch**/**EndCatch** an error occurs, otherwise execution is transferred to the first statement after **EndCatch**.

Try/**Catch**/**EndCatch** constructions cannot be nested; otherwise there can be more than one error handler per subroutine. This differs from the GFA-BASIC 16 where the Catch functions as a return from subroutine.

Example

```
test
Me.Close
```

```
Procedure test
  Local i, a%
  Try
    For i = -9 To 9
      Print @Rezip(i); ", ";
    Next i
  Catch
    Print "There is an error occured in Procedure
      test"
    Print Err$(Err)
    Print "Press any key to continue"
    KeyGet a%
  EndCatch
Return

Function Rezip(x)
  Return 1 / x
EndFunc
```

The above program will print -0.11.., -0.125, -0.14.., -0.16..,-0.2,-0.25,-0.33..,-0.5,-1, and then it will print the message and the error text, wait for a keystroke and return.

A simple change in the function, using **Try**/**Catch** again, permits to supply an error value (1/0 is not defined, but one divided by very small numbers gives a very high result, now lets supply one,catching overflows as well).

```
Function Rezip(x)
  Try
    Return 1 / x
  Catch
    Return 1E99
  EndCatch
EndFunc
```

This changed program will continue after -1 with 1E+99, 1, 0.5, 0.3.. ... And because the **Try**/**Catch** does work locally, other errors in test would be handled there.

A second example is a procedure reading a configuration value from a file

```
Local size% = 10
ReadValue("CONFIG.CFG", 1000, size%)
Print size%

Procedure ReadValue(File$, Def%, ByRef Ret%)
  Try
    Open File$ for Input As # 1
    Input # 1, Ret%
    Close # 1
  Catch
    Ret% = Def%         // Return default value
  EndCatch
  Close # 1
Return
```

A third example, just displaying a graph of the function **Sin**(x)/x. This function is defined and gives good results, except for zero, giving no result at all. Very small numbers, positive and negative, approach 1.0, so let's put this value there (**Sin**(0.0)/0.0 = 0.0/0.0 could give 1.0?).

```
Local Int a, i, y0, ys
Local ix As Double
OpenW # 1, 0, 0, _X, _Y, 0
y0 = _Y / 2, ys = _Y / 2
Color 8
For  i = 0 To _X Step 2
  ix = (i - _X / 2) / 20
  Plot i, y0 - sinx_by_x(ix) * ys
Next i
```

```
KeyGet a
CloseW 1

Function sinx_by_x(x)
  Try
    sinx_by_x = Sin(x) / x
  Catch
    sinx_by_x = 1
  EndCatch
EndFunc
```

This modified program does display a simple three dimensional view.

```
Local Int a, i, j, y0, ys
Local Double ix, jx, jx2, f, z
OpenW # 1, 0, 0, _X, _Y, 0
y0 = _Y / 2, ys = _Y / 2
Color RGB(255, 0, 0)
For j = 0 To _Y Step 4
  jx = (j - _Y / 2) / 20, jx2 = jx * jx
  For i = 0 To _X Step 2
    ix = (i - _X / 2) / 20
    f = Sqr(ix ^ 2 + jx2)
    z = y0 - sinx_by_x(f) * ys
    Pset i, z, RGB(192, 192, 192)
    Line i, z + 1, i, _Y
  Next i
  y0++
Next j
KeyGet a
CloseW 1

Function sinx_by_x(x)
  Try
    sinx_by_x = Sin(x) / x
  Catch
```

```
    sinx_by_x = 1
  EndCatch
EndFunc
```

Remarks

See **On Error** for more information on error trapping.

Known Issues

Problems can arise when using Ocx objects when an error occurs in a procedure with no Try/Catch construction which is called from another procedure with one, as shown in the example below:

```
Ocx Command cmd = "Hello", 10, 10, 100, 22
Try
  SubRoutine
Catch
  Message Err$
EndCatch
Do : Sleep : Until Me Is Nothing

Procedure SubRoutine
  Local n As Int32
  n = 2 / 0
EndProcedure

Sub cmd_Click
  Message "Hello"
EndSub
```

An error is called when the the 'Divide by Zero' error is encountered as it is captured by the main Try/Catch construct; however, the Ocx Command button is now inoperative, as would be any other Ocx objects, eventhough the program is still technically running.

To overcome this problem, simply insert a Try/Catch construct inside the called procedure as well. As can be seen if you run the amended version of the previous example below, following the error message, the Ocx Command Button is still functional and the program will continue running as designed.

```
Ocx Command cmd = "Hello", 10, 10, 100, 22
Try
  SubRoutine
Catch
  Message Err$
EndCatch
Do : Sleep : Until Me Is Nothing

Procedure SubRoutine
  Local n As Int32
  Try
    n = 2 / 0
  Catch
    Message Err$
  EndCatch
EndProcedure

Sub cmd_Click
  Message "Hello"
EndSub
```

See Also

On Error

{Created by Sjouke Hamstra; Last updated: 31/08/2015 by James Gaite}

GLL Example: AutoSave

The original GFA implementation of the auto save feature.

```
Sub Gfa_Init
  '
  ' Add menu item for AutoSave
  '
  Global Int IdxAutosave = Gfa_AddMenu("&AutoSave",
    Menu_Autosave)
  Gfa_MenuCheck(idxautosave) =
    Gfa_IntSetting("Auto_Save") And 1
  Gfa_MenuDesc(idxautosave) = "Autosave every 5
    minutes"
EndSub

Sub Menu_Autosave(idx%)
  Gfa_MenuCheck(idx) = !Gfa_MenuCheck(idx)
  Gfa_IntSetting("Auto_Save") = -Gfa_MenuCheck(Idx)
EndSub

Sub Gfa_Minute           // autosave every 5
  Minutes
  Local Date d
  Local st_old$
  // Don't save if not wanted
  If !Gfa_MenuCheck(IdxAutosave) Then Exit Sub
  // Don't save empty program
  If Gfa_LineCnt == 0 Then Exit Sub
  // Changes have bee saved before
  If !Gfa_Dirty Then Exit Sub
  // Get filetime of the autosave file
  Try
    d = FileDateTime(TempDir & "temp.g32")
  Catch
    d = 0.0
  EndCatch
  // When not older then 4.5 Minuten
  // (1 day / 24 (hours) / 60 (Minutes)) * 4.5
  // so 5 minutes or older is
  If d > Now - (1 / 24 / 60) * 4.5
    Exit Sub     //do nothing
  EndIf
  // Change statusbar
  st_old = Gfa_StatusText
  Gfa_StatusText = "Autosave as " & TempDir &
    "temp.g32"
  Gfa_SaveFile TempDir & "temp.g32"
  Gfa_StatusText = st_old
End Sub
```

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

GLL Example: Change Case

Change character case using **Gfa_Replace**. Replace the character at the cursor position to lowercase.

```
Sub Gfa_Ex_L
  Gfa_Replace Lower(Mid(Gfa_Text, Gfa_Col + 1, 1))
EndSub
```

Replace multiple characters to uppercase.

```
Sub Gfa_Ex_U
  Local Int l, c
  Exit Proc If (Gfa_SelLine != Gfa_Line)
  c = Gfa_Col
  l = Gfa_SelCol - c
  If ( l == 0)
    l = 1
  Else If l < 0
    c = Gfa_SelCol
    l = -l
    Gfa_Col = c
  EndIf
  Gfa_Replace Upper(Mid(Gfa_Text, c + 1, l))
EndSub
```

Because Gfa_Replace command doesn't cross line boundaries, the subroutine makes sure that selection to convert is at one line only. It then figures out the character to start with and length of the selection. Do not expect that the **Gfa_SelCol** is at the right of **Gfa_Col** (**Gfa_SelCol** > **Gfa_Col**).

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

GLL Example: Convert Characters

Convert characters to uppercase using Gfa_Insert.

Convert one or more characters on one or more lines to uppercase. When there is no selection the character on the right of the cursor is selected and changed to uppercase. Since **Gfa_Insert** is used to replace the selection, the conversion can cross line boundaries. **Gfa_Right** and **Gfa_Insert** are both actions that are stored in the Undo buffer, which can take 64 actions. As such**, Gfa_Ex_V** can only be undone 32 times.

```
Sub Gfa_Ex_V
  Local Int ln, c
  If Gfa_IsSelection
    Gfa_Insert Upper(Gfa_Selection)
  Else
    ln = Gfa_Line : c = Gfa_Col
    Gfa_Right
    Gfa_SelLine = ln : Gfa_SelCol = c
    Gfa_Insert Upper(Gfa_Selection)
  EndIf
EndSub
```

The difference with the previous example is that when no selection is available there is no temporary selection set as well. This reduces the number of undo actions.

```
Sub Gfa_App_U
  If Gfa_IsSelection
```

```
    Gfa_Insert Upper(Gfa_Selection)
  Else
    Gfa_Right
    If Gfa_Col
      Gfa_Replace Upper(Mid(Gfa_Text, Gfa_Col - 1,
        1))
    EndIf
  EndIf
EndSub
```

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

GLL Example: Using Eval()

Evaluate an expression using Eval().

Input from inside a GLL routine can be performed with InputBox() or Prompt. This example uses the GFA-BASIC 32-Function **Eval** to calculate a mathematical expression. The result is then displayed in a message box.

```
Sub Gfa_Ex_Y     ' SHIFT+CTRL+Y
  Local a$, x#
  a = InputBox("Give an expression")
  If Len(a)
    //Try
    x = Eval(a)
    MsgBox "The result is" & x
    //Catch
    //  MsgBox "error"
    //EndCatch
  EndIf
EndSub
```

Rather than use an input box you could use selected text and then replace the selection with the evaluation result

```
' SHIFT+CTRL+Z

Sub Gfa_Ex_Z
  Try
    Gfa_Insert Eval(Gfa_Selection)
  Catch
  EndCatch
EndSub
```

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

GLL Example: Insert Snippet Code

The following, often used, snippet is required to create a program that uses a form created in the form editor. The default name of the form is frm1.

```
LoadForm frm1 Hidden
Do
  Sleep
Until Me Is Nothing

Sub frm1_Load
  /* Todo: initialise controls */
  frm1.Show
EndSub
```

Additionally, code for the event Sub frm1_Load is attached to perform initialization of the controls the form contains. To prevent flicker the form is loaded invisible and displayed after the initializations.

```
Sub Gfa_Init
  Gfa_AddMenu "Load Form Snippet",
    Gfa_Menu_CreateBasic
End Sub

Sub Gfa_Menu_CreateBasic(i%)
  Local a$
  a = InputBox("Enter name of form", "", "frm1")
  If Len(a)
    ' Insert main program
    Gfa_Insert #10#10
    Gfa_Insert "LoadForm " & a & " Hidden"#10#10
    Gfa_Insert "Do" #10 "Sleep" #10 "Until Me Is
      Nothing"
    Gfa_Insert #10#10
    ' Insert _Load event sub
    Gfa_Line = Gfa_LineCnt
    Gfa_Insert #10#10 "Sub " & a & "_Load" #10
    Gfa_Insert "/* Todo: initialize controls " +
      "#10#10"
    Gfa_Insert a & ".Show"#10#10
    Gfa_Insert "End Sub"#10
  EndIf
End Sub
```

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

GLL Example: Add a Resource

Add a resource to :Files section using Gfa_CopyFile.

This example shows a way to add a resource file to the inline section on the :Files tab. It assumes you put the filename of the resource in the source code in the following format.

```
//:icodeb = e:\cparse\icodeb.ico
// The GLL sub (Shift + Ctrl + C) to handle the
  copy process is here.

Sub Gfa_Ex_C
  Local a$, b$, i%
  a = Trim(Gfa_Text)
  If Left(a, 3) != "//:" Then Exit Sub
  a = Mid(a, 3)
  i = InStr(a, "=")
  If !i Then Exit Sub
  b = LTrim(Mid(a, i + 1))
  a = RTrim(Left(a, i - 1))
  If Len(b) == 0 || Len(a) <= 1 Then Exit Sub
  If Exist(a)
    MsgBox a & " exists"
    Exit Sub
  EndIf
  Try
    Gfa_CopyFile b, a
  Catch
    MsgBox "Error while copying"#10"from " & b & "
      to " & a
  EndCatch
```

```
End Sub
```

When the cursor is located in the line starting with a //: comment, the comment is analyzed to look for a resource to be added to the inline section. The name of the inline resource is specified with the string starting with the colon (:icon1). Then the line searched for the name of the file to load, which must be preceded with ' = '. If the file doesn't exist a message is displayed, otherwise the file is copied to the inline section using **Gfa_CopyFile**.

This example can be extended by overriding the previous inline entry. However, the inline resource must de deleted first. To delete the :File use **Gfa_CopyFile ""**, when **Exist**(a) is true.

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

GLL Example: Jump to subroutine

Jump between procedure headers quickly.

```
Sub GotoProcHeader(Optional Direction As Int =
  CurrentProcHeader)
  Global Enum CurrentProcHeader,
    PreviousProcHeader, NextProcHeader,
    FirstProcHeader, LastProcHeader
  Local Int i
  Switch Direction
  Case CurrentProcHeader
    Gfa_Line = Gfa_ProcLine
  Case PreviousProcHeader
    Gfa_Up
    If Gfa_Proc <> ""
      Gfa_Line = Gfa_ProcLine
    Else
      Gfa_Down
      Gfa_StatusText = "Reached first procedure."
    EndIf
  Case NextProcHeader
    Local String curProc = Gfa_Proc
    i = Gfa_Line + 1
    While i < Gfa_LineCnt
      Gfa_Line = i
      If Gfa_Proc <> curProc
        Gfa_Line = Gfa_ProcLine
        Exit Do
      EndIf
      i ++
    Wend
    If i = Gfa_LineCnt Then
      Gfa_Line = Gfa_ProcLine
      Gfa_StatusText = "Reached last procedure."
    EndIf
  Case FirstProcHeader
    Gfa_Line = 1
    GotoProcHeader (NextProcHeader)
  Case LastProcHeader
    Gfa_Line = Gfa_LineCnt
    Gfa_Line = Gfa_ProcLine
  EndSwitch
End Sub
```

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

IsArray Function

Purpose

Returns a **Boolean** value indicating whether a variant holds an array.

Syntax

Bool = **IsArray**(varname)

Description

IsArray returns True if the variable is an array; otherwise, it returns False. **IsArray** is especially useful with variants containing arrays.

Example

```
Local a As Variant, x
a = Array(1, 2, 3)
Print a(1)
Print IsArray(a)
Local b%(10)
Print IsArray(b%(1))
```

See Also

Array, IsDate, IsEmpty, IsError, IsMissing, IsNull, IsNumeric, IsObject

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Array Function

Purpose

Returns a **Variant** SafeArray containing a one-dimensional array.

Syntax

variant = **Array**([parameter list]) [As Type]

parameter : any expression

Description

The required parameter list argument is a comma-delimited list of values that are assigned to the elements of the array contained within the Variant. If no arguments are specified, an array of zero length is created. **Note:** The lower bound of an array created using the **Array** function is always 0, regardless of the value of **Option Base**.

In addition, the varType of the array can be specified using the optional **As** *Type* statement: hence, a ByteArray is created by adding `As Byte` after the parenthesized parameter list.

Finally, the boundaries of the array can be returned using **LBound** (always 0) and **UBound**; however, bar those listed in this section, no other Array-specific commands and functions work with this type of array.

Example

```
Debug.Show
Global V As Variant = Array(1, 2, 3, "GFA32")
Dim Va As Variant
Va = Array(1, 2, Array(15, 20) As Byte)
// To refer to an element of the array in the
  array:
Trace V(2)              // output: 3
Trace IsArray(Va(2))    // output: True
Trace Va(2)(1)          // output: 20
Trace TypeName(Va(2))   // output: Byte()
```

Known Issues

GFA-BASIC32 support for dynamic arrays inside Variants is very poor.

Unlike VB and VB.NET, it does not support **ReDim** on a Variant containing an array. The only way to resize an array is to re-use the **Array** function as below:

```
Local v2 As Variant = Array(1, 15, "Last one")
Print v2(0), v2(1), v2(2)
v2 = Array(v2(0), v2(1), v2(2), "And now this
  one")
Print v2(0), v2(1), v2(2), v2(3)
```

It also only allows elements to be amended if the index is a constant not a variable:

```
Local v2 As Variant = Array(10, 12, 108, "Hello")
Local Const elem = 2 : Local Int32 elem% = 2
v2(2) = 20 : Print v2(2)            // Prints 20
v2(elem) = 30 : Print v2(elem)      // Prints 30
v2(elem%) = 40 : Print v2(elem%)    // Throws an
  'Object is Nothing' error
```

[Reported by James Gaite. 22/02/17]

Another problem with trying to set values in this type of array is found when the element is part of an array *within* another array. When typing in `array(1)(2) = "Hello"` for example, the error 'No compare string <=> number' is thrown; alternatively, if you try passing a numerical value auch as 5 to the same element, the editor rewrites the line as `array(1), (2) = 5` and the message 'Error 0x80004003 Invalid pointer' is displayed.

The only way to get around this is to 'step' up to the desired array in which you want to make the change, alter it, then 'step' down again to reset the parent array, something like this:

```
Local v2 As Variant = Array(10, 12, 108, Array(5,
  "Hello"))
Local Variant a, b
a = v2(3)
a(0) = "Goodbye"
v2(3) = a
Print v2(3)(0)
```

[Reported by James Gaite. 22/02/17]

See Also

IsArray(), Array ()=

{Created by Sjouke Hamstra; Last updated: 08/03/2018 by James Gaite}

Option Base Command

Purpose

Sets the starting offset for row/column indexing of arrays.

Sets the starting offset for random I/O files to 0 or 1.

Syntax

Option Base [an] [, rb]

an, rb:iexp

Description

The **Option Base** *an* command sets the starting offset for row/column indexing of arrays. In case of **Option Base** 0 the indexing starts with element 0, and in case of **Option Base** 1 with element 1. **Option Base** 0 is the default.

Option Base ,*rb* will set the default base for random I/O files to 0 or 1.

A note of caution: if an array is dimensioned under **Option Base** 0 and then **ReDim**'ed under **Option Base** 1, the array retains its original starting element of 0; the same happens the other way around.

Example

```
Option Base 1
Print "Option Base is"; CheckOptionBase
Option Base 0
```

```
Print "Option Base is"; CheckOptionBase

Function  CheckOptionBase
  Local a%(2), rv% = LBound(a%())
  Return rv%
EndFunction
```

Remarks

Option Base also sets the starting index of the **Mat** commands.

Known Issues

There are issue when **Option Base 1** is set with a number of different commands, including ReDim and Array()=. See the relevant pages for more information.

In addition, you may find problems when using **Option Base 1** when passing arrays using **ByRef** to another procedure and, on occasions, if the array was not created using **Dim** (i.e. the array was dimensioned using just **Local** or **Global**): in both these instances, an element '0' (zero) may be added. However, this error does not happen all the time and is one of the few 'random' errors that occur in GB32 from time to time. If these problem arise, you can generally get around them by either adding **Dim** to the original declaration or creating a local array within the procedure to which the array is to be passed and copying the global array into it.

See Also

Open

{Created by Sjouke Hamstra; Last updated: 17/12/2015 by James Gaite}

Dim Command

Purpose

declares a variable of any kind.

Syntax

[**Local** | **Global** | **Static**] **Dim** *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*] [, *varname*[**(**[*subscripts*]**)**]] [**As** [**New**] *type*]

Description

In GFA-BASIC 32 it is mandatory to declare variables before they can be used. **Dim** is the general command to declare a variable. Others are **Global**, **Local**, and **Static**. A variable declared with **Dim** has local scope inside a subroutine and global scope when it is declared in the main part of the program. Optionally, **Global**, **Local**, and **Static** may be used together with **Dim**, but when used these commands may do without the **Dim** part. The following is allowed:

Dim a%, b&, s$, v

Global Dim a As Long, b As Short, s As String, v As Variant

Global a As Long, b As Short, s As String, v

If you don't specify a data type or object type, and there is no **Def***type* statement, the variable is **Variant** by default.

Arrays can have up to 7 dimensions. The *subscripts* argument uses the following syntax:

[*lower* **To**] *upper* [, [*lower* **..**] *upper*] **. . .**

When not explicitly stated in *lower*, the lower bound of an array is controlled by the **Option Base** statement. The lower bound is zero if no **Option Base** statement is present.

You can also use the **Dim** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to re-declare a dimension for an array variable whose size was explicitly specified in a **Dim** statement, an error occurs.

If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it, so you don't have to use the **Set** statement to assign the object reference. The **New** keyword can be used only on the following object types: **Picture**, **Font**, **StdPicture**, **StdFont**, **Collection**, **DisAsm**, **CommDlg** and **ImageList**.

Example

```
Dim Names(9)                  ' Declare an array
 with 10 elements.
Dim Names(1 To 9)             ' Declare an array
 with 9 elements.
Dim Names(0 .. 9, 0 .. 1)     ' Declare an array
 with 2 dimensions.
Dim Names()                   ' Declare a dynamic
 array
Dim MyVar, MyNum              ' Declare two
 variables
Dim dis As New DisAsm         ' Declare a new
 instance of the object
```

Remarks

Allowed are Types in arrays, arrays in Types, or Ocx-array‘s.

A local array doesn't need to be erased (**Erase**) at the end of a subroutine, this is done automatically.

If an array is dimensioned under **Option Base** 0 and then **ReDim**'ed under **Option Base** 1, the array retains its original starting element of 0; the same happens the other way around.

See Also

Global, Local, Static, Dim(), IndexCount, LBound, UBound, Erase, ReDim, Clr

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 30/09/2014 by James Gaite}

Private and Public commands

Purpose

Used to declare local and global variables and constants.

Syntax

Private [**Dim**] [**Const** | **Enum**] varname [**As** Type] [,..]

Public [**Dim**] [**Const** | **Enum**] varname [**As** Type] [,..]

Description

Private and **Public** are implemented to make porting VB code easier. In fact, **Public** is the same as **Global** and **Private** is the same as **Local**.

NOTE: GFABasic only accepts **Private** and **Public** in relation to variables and constants, not declared APIs. If the latter are imported, the IDE will flag them up as errors.

Remarks

In VB subroutines are declared **Private** or **Public** often. VB divides the source code about multiple files and need to know which procedures are to be used inside the file only, or outside the file as well. Hence the keywords **Private** and **Public**. In GFA-BASIC 32 all subroutines are equal and cannot be qualified as either local or global. When porting VB code to GFA-BASIC 32, you must remove these qualifiers from the code.

Known Issues

When using local private arrays, you may develop a memory leak problem. This stems from the fact that the compiler forgets to add destruction code for local arrays when an explicit local declaration of a string variable is absent. As a workaround, in any procedure, function or sub which declares a local array, add a local string variable dummy$ if none other is present.

See Also

Dim, Global, Local, Static, Const, Enum

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

LBound, UBound and IndexCount Functions

Purpose

Returns details of either the indicated dimension or the whole of an array.

Syntax

LBound[(] *array*[, *dimension*] [)]

UBound[(] *array*[, *dimension*] [)]

IndexCount[(] array() [)]

array	*: varname*
dimension	*: iexp*

Description

The **LBound** function is used with the **UBound** function to determine the size of an array. Use the **UBound** function to find the upper limit and **LBound** the lower limit of an array dimension.

By default, the lower bound for any dimension is always 0. This can be changed using **Option Base** n.

dimension is a whole number indicating which dimension's lower bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If *dimension* is omitted, 1 is assumed.

IndexCount() returns the number of array dimensions.

Example

```
Option Base 1
Debug.Show
Dim MyArray(1 To 10, 5 To 15, 10 To 20)
Dim AnyArray(10)
Trace IndexCount(MyArray())
Trace LBound(MyArray(), 1)
Trace UBound(MyArray(), 1)
Trace LBound MyArray(), 2        // All three
  functions can be entered ...
Trace UBound MyArray(), 2        // ...without the
  brackets enclosing the parameters
Trace LBound(MyArray(), 3)
Trace UBound(MyArray(), 3)
Trace IndexCount MyArray()
Debug.Print
Trace IndexCount(AnyArray())
Trace LBound(AnyArray())         // Returns 0 or 1,
  depending on setting of Option Base.
Trace UBound(AnyArray())
```

Remarks

See Also

Dim?

{Created by Sjouke Hamstra; Last updated: 17/05/2017 by James Gaite}

Dim? Function

Purpose

Returns the total number of elements in an array.

Syntax

Dim? [(] x() [)]

x() : array of any variable type

Example

```
OpenW # 1
Dim a%(19, 9, 2, 13)
Print Dim?(a%())         // Prints 8400
Print IndexCount(a%())   // Prints 4
Print ArraySize(a%())    // Prints 33600
Do
  Sleep
Until Me Is Nothing
```

See Also

Dim, Erase, LBound, UBound, IndexCount, ArraySize

{Created by Sjouke Hamstra; Last updated: 17/05/2017 by James Gaite}

Erase Command

Purpose

Deletes all arrays listed after it.

Syntax

Erase x1() [,x2(),...]

x1(),x2(),...:arrays of any type

Description

The arrays in the list after **Erase** must be separated by commas.

Example

```
OpenW # 1
Dim a#(5), b%(3), i%
ArrayFill a(), PI
ArrayFill b%(), 42
Mat Print a()
Print
For i% = 1 To 3
  Print b%(i%)
Next i%
Erase a(), b%()
Try
  Print b%(2)     // Array bounds exceed error
Catch
  Print "Array Bounds Error - b%(2) no longer
    exists"
```

```
EndCatch
```

Remarks

Erase clears all elements of the array from memory but does not delete the array reference; this is cleared for local variables when the procedure is exited, and for global variables when the program ends. For this reason, **Erase** can be used to clear all values from an array before **ReDim**-ing it.

See Also

Clr, Dim, Redim

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

Insert Command

Purpose

Inserts a numeric or a string expression at the specified place in a one-dimensional array of corresponding variable type.

Syntax

Insert x(m) [=y]

y:aexp, if x() is a numeric array or sexp, if x() is a string array
x():a one-dimensional array of any type

Description

Insert x(m)=y inserts y in array x() at position m. In other words all items in array x() whose indices are greater than or equal to m are moved one position down. The last element in x() is deleted with each **Insert**.

Insert x(m) inserts an empty element at position m.

Example

```
OpenW # 1
Dim a$(4), i%
a$(1) = "String #1"
a$(2) = "String #2"
a$(3) = "String #3"
a$(4) = "String #4"
Insert a$(3) = "New String"
```

```
For i% = 1 To 4
  Print a$(i%)
Next i%
// prints
// String #1
// String #2
// New String
// String #3
```

See Also

Delete

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Delete Command

Purpose

deletes an element from a one-dimensional array of any variable type.

Syntax

Delete x(m)

m:integer expression

x():one-dimensional array of any variable type

Description

Delete x(m) deletes the element indexed by m from the array x(). In other words all array items whose indices are greater than or equal to m are shifted one position up. The last element in the array is deleted (with 0 or "" depending on type).

Example

```
OpenW # 1
Dim a$(4), i%
a$(1) = "Text 1"
a$(2) = "Text 2"
a$(3) = "Text 3"
a$(4) = "Text 4"
Delete a$(3)
For i% = 1 To 4
  Print a$(i%)
```

```
Next i%
// prints
// Text 1
// Text 2
// Text 4
```

See Also

Insert

{Created by Sjouke Hamstra; Last updated: 30/09/2014 by James Gaite}

ArrayAddr Function

Purpose

Returns the memory address of an array

Syntax

x = **ArrayAddr**[(] array() [)]

x : avar

Description

ArrayAddr() returns the memory location of the first byte of the array data. (Not string array.)

Example

```
Debug.Show
Dim a|(100), b$(4)
Trace ArrayAddr(a|())
Trace V:a|(0)
b$(0) = "GFA"
Trace ArrayAddr b$()
Trace V:b$(0)
```

See Also

ArraySize()

{Created by Sjouke Hamstra; Last updated: 17/05/2017 by James Gaite}

ArraySize Function

Purpose

Returns the size of the array in bytes

Syntax

x = **ArraySize**[(] array() [)]

Description

ArraySize returns the number bytes occupied by the entire array (not string array).

Example

```
Debug.Show
Dim a(10), b%(3, 4), c&(2, 3, 4)
Dim d|(1, 2, 3, 4), e!(5)
Trace ArraySize(a())    // prints 176
Trace ArraySize b%()    // prints 80
Trace ArraySize(c&())   // prints 120
Trace ArraySize d|()    // prints 120
Trace ArraySize(e!())   // prints 24
```

See Also

ArrayAddr(), ArrayFill

{Created by Sjouke Hamstra; Last updated: 17/05/2017 by James Gaite}

ArrayFill Command

Purpose

Initializes a numerical array of any type with a value.

Syntax

ArrayFill a(),x

a():any numeric or Boolean array

x:aexp

Description

The **ArrayFill** a(),x command can be used on all numeric and Boolean arrays. The complete array a(), including all dimensions, is filled with the expression x. By default, all dimensioned numeric arrays are cleared with 0, while all Boolean arrays are initialized with False - which is also 0.

Example

```
OpenW # 1
Dim a(10), b%(3, 4), c%(2, 3, 4), d|(1, 2, 3, 4),
  e!(5)
ArrayFill a(), 17.4
ArrayFill b%(), 13
ArrayFill c%(), 17
ArrayFill d|(), 9
ArrayFill e!(), True
Print a(1)     // Prints 17.4
```

See Also

Mat Set

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

QSort Function

Purpose

Sorts the elements in an array by its size using the Quicksort algorithm.

Syntax

QSort x(s) [,n] [,m%()]

QSort x$(s) [Compare c][**With** n()] [,n [,m%()]]

s	*: + or - for ascending or descending order*
c, n	*: integer expression*
x()	*: one dimensional floating point or integer array*
x$()	*: one dimensional sting array*
n()	*: one dimensional integer array with 8-, 16- or 32-bit integer variables*
m%()	*: one dimensional integer array with 32-bit integer variables*

Description

The s enclosed in round brackets can be replaced with a "+", a "-" or may be left out. "+" or no specification results in arrays x() and m%() being sorted in ascending order. In this case, after the sorting, the smallest array element assumes the smallest index (0 for Option Base 0 or 1 for Option Base 1). "-" results in the array being sorted in descending order. In this case, after the sorting, the biggest array element assumes the smallest index.

The parameter n specifies that only the first n elements of the array should be sorted. For Option Base 0 these are the elements with indices 0 to "n"-1, and for Option Base 1 the elements with indices 1 to "n". If n is given explicitly, it can be followed by a Long integer array, which will be sorted together with the array x(), that is to say, each swap in array x() will also be performed in array m%(). This is particularly useful when the sorted array x() contains the sort key (for example the postal code), while other arrays contain additional information that must maintain the same order as the keys.

When sorting string arrays (x$()) a sort criterion can be specified with **With**, in the form of an array n() with at least 256 elements. If **With** is not given the normal ASCII table is used as the sort criterion.

Another option when sorting string arrays is to use **Compare** *c*: this allows you to specify different comparison methods locally - case sensitive or insensitive, the sorting of accented characters, etc. - without changing the global Mode Comapre setting. The comparison method is determined by the value entered in *c* which corresponds to the values used with **Mode Compare** so for a case sensitive search *c* = -1, for a case insensitive search with correct sorting of accented characters *c* = 1, and so on.

Examples

This first example shows a sort with a dependant array:

```
Local i%, n% = 3
Dim a(n%), b%(n%), c$(n%), d$(n%)
Restore m1 : For i% = 0 To n% : Read a(i%) : Next
  i%
m1: : Data 10,-3,5,21
```

```
Restore m2 : For i% = 0 To n% : Read c$(i%) : b%
  (i%) = i% : Next i%
m2: : Data A,B,C,D
Restore m3 : For i% = 0 To n% : Read d$(i%) : Next
  i%
m3: : Data Who,How,What,Where
OpenW # 1 : GraphMode , TRANSPARENT :
  Win_1.FontName = "Courier"
For i% = 0 To n%
  Print Str$(a(i%), 5, 2)``
  Print Str$(b%(i%), 5, 2)``
  Print c$(i%)``
  Print d$(i%)
Next i%
Print
QSort a(), n%, b%()
For i% = 0 To n%
  Print Str$(a(i%), 5, 2)``
  Print Str$(b%(i%), 5, 2)``
  Print c$(b%(i%))``
  Print d$(b%(i%))
Next i%
```

Prints first of all (unsorted)

```
10.00 1.00 A Who
-3.00 2.00 B How
 5.00 3.00 C What
20.00 4.00 D Where
```

then (sorted)

```
-3.00 2.00 B How
 5.00 3.00 C What
10.00 1.00 A Who
20.00 4.00 D Where
```

This second example shows a sort using the **With** keyword:

```
// Create array to be sorted
Local a$() : Array a$() =
  "D"#10"H"#10"A"#10"z"#10"c"
// RUN 1: populate the With array d%() with
  character codes in descending order
Local d%(255), n% : For n% = 0 To 255 : d%(n%) =
  255 - n% : Next n%
// Show the array to be sorted before the sort
For n% = 0 To 4 : Print a$(n%), : Next n% : Print
// Sort in descending order which, as d%() is also
  descending will result in an ascending sort by
  ANSI code
QSort a$(-) With d%()
// The result of the sort
For n% = 0 To 4 : Print a$(n%), : Next n% : Print
// RUN 2: populate the With array d%() with a
  'Text' sort to ignore capital letters
Local b$ = " !" & #34 & "#$%&'()*+,-./0123456789:;
  <=>?
  @AaÀàÁáÂâÃãÄäÅåÆæBbCcÇçDdÐðEeÈèÉéÊêËëFfGgHhIiÌìÍí
  ÎîÏïJjKkLlMmNnÑñOoÒòÓóÔôÕõÖöØøŒœPpQqRrSsßŠšTt" &

  _
  "UuÙùÚúÛûÜüVvWwXxYyÝýŸÿÞþZzŽž[\]^_`{|}~€�‚ƒ„…
   †‡ˆ‰‹ ���‘’“”•–—™›� ¡¢£¤¥¦§¨©ª«¬®¯°±²³
   ´µ¶·¸¹º»¼½¾¿×÷"
For n% = 0 To 31 : d%(n%) = n% : Next n%
For n% = 32 To 255 : d%(n%) = Asc(Mid(b$, n% - 31,
  1)) : Next n%
// Sort the array in ascending order
QSort a$(+) With d%(), 4 // <--- Without the
  'count' variable, this does not work properly.
// And print the result
For n% = 0 To 4 : Print a$(n%), : Next n% : Print
```

The final example illustrates how to use of the **Compare** keyword:

```
Local Int32 m%(10), n%, x$(10)
For n% = 0 To 10 : m%(n%) = 10 - n% : Read x$(n%)
  : Next n%
// Descending Case Sensitive sort of the first 7
  elements only
QSort x$(-) Compare -1, 7, m%()
For n% = 0 To 10 : Print n%, m%(n%), x$(n%)  :
  Next n%
Data "A","c","D","e","f","J","K","m","N","p","Z"
```

Remarks

Interestingly, running a sort with dependant array to be sorted at the same time seems to be quicker than if the the second array is omitted as the following code snippet shows:

```
Dim a$(120), a%(120), n%, t#
For n% = 1 To 120 : a$(n%) = Chr(65 + (Rnd * 26))
  : a%(n%) = n% : Next n
t# = Timer
For n% = 1 To 10000
  QSort a$()
Next n%
Print Timer - t#
t# = Timer
For n% = 1 To 10000
  QSort a$(), 120, a%()
Next n%
Print Timer - t#
```

[Reported by James Gaite. 24/01/17]

Known Issues

Using the **With** keyword can sometimes lead to inaccurate results: to see this, copy the second of the two examples and remove the count variable from the second **QSort** statement - the 'z' will now be ordered as the second character, not the last. Then replace the count variable and change the 'H' in the array to be sorted to 'h'; now 'h' is the second character listed.

A second problem can occur when using **With** if not all of the 256 ANSI values are included - this leads to results similar to those highlighted above.

General advice is not to use the **With** keyword unless necesary and use the **Mode Compare** settings where possible in its place.
[Reported by James Gaite. 16/01/17 & 26/02/17]

If you sort a string array in descending order where all the strings are null or blank(""), then an Access-Violation Exception error message will be thrown. This does not happen if the sort is in ascending order.
[Reported by James Gaite. 17/01/17]

{Created by Sjouke Hamstra; Last updated: 27/01/2019 by James Gaite}

Store and Recall Commands

Purpose

Fast save and load of text files.

Syntax

Store #n, a$() [,m]

Recall #n, a$(),m,j

n:integer expression; channel number
a$():one dimensional string array
m:iexp
j:ivar

Description

Store saves the complete string array through the opened channel n (from 0 to 511) to a file. The individual strings in the file saved with **Store** are separated by a CR/LF. The parameter m is optional and defines how many strings from a$() should be written to the text file.

Recall #n,a$(),m,j reads through an already opened channel n (from 0 to 511) m lines from a text file, into the string array a$(). If m is greater than the number of elements in the string array, the number of reads is automatically limited (m=-1 fills the whole array). If during reading an **EOF** is reached the reading is stopped without reporting an error. At the end of the read the variable j contains the number of strings actually read in.

Recall expects that the single character strings are separated by CR/LF within the text file. If the text file follows this structure, Recall also can be applied to files which haven't been produced with **Store**. It should also be noted that, although **Store** can save string elements of any legal length, **Recall** can only retrieve upto 9999 characters - see below.

Example

```
' Create a text file
Local a$(6999), i%, x%
Local fn$ = App.Path + "\Test.txt"
For i% = LBound(a$()) To UBound(a$())
  a$(i%) = "Hello world" & Str(i% + 1)
Next i%
Open fn$ for Output As # 1
Store # 1, a$(), UBound(a$()) + 1
Close # 1
' Load text file
' Erase a$() : Dim a$(10000) // This causes an
  error as Recall can not recognise a$, so use the
  following...
ReDim a$(10000)
Try
  Open fn$ for Input As # 1
  Recall # 1, a$(), -1, x%
  Close # 1
  Message "Lines read in by Recall: " & Str( x%)
  OpenW 1
  Ocx ListBox lb1 = , 50, 50, _X - 100, _Y - 100
  lb1.Sorted = False
  For i% = LBound(a$()) To UBound(a$())
    If a$(i%) <> "" Then lb1.AddItem a$(i%)
  Next i%
  Do
    Sleep
  Until Me Is Nothing
Catch
  Message Err.Description
EndCatch
Kill App.Path + "\Test.txt"
```

Reads the complete text file TEST.TXT in the local folder into the string array a$() and, when finished, prints how many strings have been read in and then loads all elements into a listbox.

Remarks

Store and **Recall** are for strings what **BSave**/**BLoad** or **BPut**/**Bget** are for general arrays and memory areas.

Known Issues

1. As shown in the example above, you can not use an array that has been **Erase**-d in a **Recall** statement as GFA reads the array as null, does not import any values and returns an error if you try and interrogate the array. Rather than **Erase**, use **Redim** instead.

---

2. As explained in the Description, **Recall** can only retrieve 9999 bytes for each string element; the remainder of the bytes (or the next 9999) it places in the next string element, nudging all the other elements up one. This is illustrated by the code below:

```
Local a$(100), n As Int32
For n = 1 To 100 : a$(n) = String(12000, "A") :
  Next n
Open App.Path & "\store.tmp" for Output As # 1
Store # 1, a$()
```

```
Close # 1
For n = 1 To 100 : a$(n) = "" : Next n
Open App.Path & "\store.tmp" for Input As # 1
Recall # 1, a$(), -1, n
Close # 1
Print Len(a$(1))
Print Len(a$(2))
Kill App.Path & "\store.tmp"
```

To fix this problem, use the *RecallX* function included in the example below. Also note the *StoreX* procedure which, even though it is not used in this example, allows you to use customised element dividers.

```
// Note: Unlike Recall, RecallX() has an optional
  parameter which,...
// ...if set, redimensions the string array to fit
  all requested...
// ...elements if the original string is too small
// Both StoreX() and RecallX() allow for a
  customised element divider
Option Base 0
Local a$(10), n As Int32
For n = 1 To 10 : a$(n) = String(120000, "A") :
  Next n
Open App.Path & "\store.tmp" for Output As # 1
Store # 1, a$()
Store # 1, a$()
Close # 1
ReDim a$(5)
Open App.Path & "\store.tmp" for Input As # 1
Print "No of elements: "; RecallX(1, a$(), 5, n)
For n = 1 To UBound(a$()) : Print Len(a$(n)) :
  Next n
Print "No of elements: "; RecallX(1, a$(), -1, n,
  True)
```

```
For n = 1 To UBound(a$()) : Print Len(a$(n)) :
  Next n
Close # 1

Procedure StoreX(filenumber%, ByRef a$(), Optional
  ct%= -1, Optional elemdiv As Variant)
  // Store works exactly as required
  // This replacement is only included for if you
    wish to change the end markers...
  // ...of each string from CRLF to something else
    to allow Store/Recall...
  // ...to work with string elements which contain
    CRLF markers for line breaks...
  // ...or when the array is used to save picture
    and/or file information which...
  // ...may have the pairing #13#10 numerous times
    within one element.
  Dim b$(1), c$, ed$, ob| = LBound(b$()), n%
  // ob| is used to adjust for Option Base
  ed$ = Iif(IsMissing(elemdiv), #13#10, elemdiv)
  If ct% <> -1 : If ob| = 0 Then Inc ct%
  Else : ct% = UBound(a$()) - Iif(ob| = 0, 1, 0)
  EndIf
  For n = ob| To ct%
    c$ = a$(n) & ed$ : BPut # 1, V:c$, Len(c$)
  Next n
EndProcedure

Function  RecallX(filenumber%, ByRef a$(), ct%,
  ByRef retval%, Optional redim?, Optional elemdiv
  As Variant)
  // Inspired by an example by Roger Cabo
  Local b$, bsize%, c$, ed$, ended?, p1%, rec%,
    redimmed?
  ed$ = Iif(IsMissing(elemdiv), #13#10, elemdiv)
  // If End of File or file is empty, return
    everything as it was
```

```
If EOF(# filenumber%) Then Return 0
// Set Start Element to match Option Base
Dim t%(1) : rec% = LBound(t%()) : retval% = 1
// Clear First Element of Array
a$(rec%) = ""
// Start Retrieving file date
While Not EOF(# filenumber%) And Not ended?
  // Calculate size of block to retrieve
  bsize% = Min(64000, LOF(# filenumber) - Loc(#
    filenumber))
  // Retrieve String block
  b$ = Input$(bsize%, # filenumber%)
  // Search for CRLF...
  p1% = InStr(b$, ed$)
  // ...and split records when found
  While p1% <> 0 And Not ended?
    a$(rec%) = a$(rec%) & Left(b$, p1% - 1) : b$ =
      Mid(b$, p1% + Len(ed$))
    // If reached predetermined file limit then
      end...
    If ct% <> -1 And retval% = ct% : ended? = True
    Else // ...otherwise increase a$() and carry
      on
      If rec% = UBound(a$()) And redim? : ReDim
        a$(UBound(a$()) + 100) : redimmed? = True
        Inc rec% : Inc retval% : a$(rec) = ""
      ElseIf rec% = UBound(a$()) : ended? = True
      Else : Inc rec% : Inc retval% : a$(rec) = ""
      EndIf
    EndIf
    p1% = InStr(b$, ed$)
  Wend
  // If End of File reached, add what remains of
    b$ to the last a$() record...
  If Not ended? : a$(rec%) = a$(rec%) & b$
    // ...otherwise, b$ may be part of another
      data block to move file pointer back
    Else : Seek # filenumber%, Loc(# 1) - Len(b$)
    EndIf
  Wend
  If Not ended? Then Dec retval% : Dec rec%
  If redimmed? Then ReDim a$(rec%)
  Return retval%
EndFunction
```

See Also

Open

{Created by Sjouke Hamstra; Last updated: 16/07/2015 by James Gaite}

GFABASIC 32 Language Reference

This information title contains reference material on the GFA-BASIC 32 language:

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Bclr Function

Purpose

clears one bit in an integer expression.

Syntax

i = **Bclr**(m, n) (function)

Bclr v, n (command)

m, niexp
v:ivar

Description

Bclr(m, n) clears the n-th bit in the integer expression m (the bit is set to 0) and returns a 32-bit integer.

Bclr ivar, n clears the n-th bit in an integer variable.

Example

```
OpenW # 1
Dim i% = 11                             //  11 =>
  1011
i% = Bclr(i%, 0) : Print Bin$(i%, 4)   // Prints
  1010
Bclr i%, 1 : Print Bin$(i%, 4)         // Prints
  1000
```

See Also

Bset(), Btst(), Bchg()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Bset Function

Purpose

Sets one bit in an integer expression or variable.

Syntax

I = **Bset**(m, n)

Bset ivar, n

m, n:iexp
ivar:ivar

Description

Bset(m, n) sets the n-th bit in the integer expression m and returns the new value.

Example

```
OpenW # 1
Dim i% = 10                               //  10 =>
  1010
i% = Bset(i%, 0) : Print Bin$(i%, 4)     // Prints
  1011
Bset i%, 2 : Print Bin$(i%, 4)           // Prints
  1111
```

See Also

Bclr(), Btst(), Bchg()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Btst Function

Purpose

Tests the bit status in an integer expression.

Syntax

Bool = **Btst**(m, n)

m, n:iexp

Description

Btst(m, n) returns -1(true) when the n-th bit in the integer expression m is set, and 0 (false) if it's not.

Example

```
Local i% = 10      // 10 => 1010
Print Btst(i%, 3) // Prints True
```

See Also

Bclr(), Bset(), Bchg()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Bclr8 Function

Purpose

Clears one bit in a 64-bit integer expression.

Syntax

i64 = **Bclr8**(m, n)(function)

~~**Bclr8** i64var, n(command)~~

m : i64var

n : iexp

Description

Bclr8(m, n) clears the nth bit of a 64-bit integer m and returns a 64-bit value.

~~**Bclr8** i64var,n clears the nth bit of a 64-bit variable.~~

Example

```
OpenW # 1
Dim i64 As Large = 11                        //  11
  => 1011
i64 = Bclr8(i64, 0) : Print Bin$(i64, 4)   //
  Prints 1010
```

Remarks

Although listed in the original help file as a command - i.e. Bclr8 v64, n - it seems never to have been implemented as such.

See Also

Bset8, Btst8, Bchg8

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Bset8 Function

Purpose

Sets one bit in an integer expression or variable.

Syntax

Large = **Bset8**(m, n)

~~**Bset8** var64, n~~

n:iexp m,

var64:int64 exp

Description

Bset8(m, n) function sets the n-th bit in the integer expression m and returns the new 64 bit integer value.

~~**Bset8** var64,n command sets the n-th bit in the integer variable var64.~~

Example

```
OpenW # 1
Dim i64 As Large = 10                     //  10
  => 1010
i64 = Bset8(i64, 0) : Print Bin$(i64, 4)  //
  Prints 1011
```

Remarks

Although listed in the original help file as a command - i.e. Bset8 v64, n - it seems never to have been implemented as such.

See Also

Bclr8(), Btst8(), Bchg8()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Btst8 Function

Purpose

Tests the bit status in a 64-bit integer expression.

Syntax

Bool = **Btst8**(i64,n)

i64:int64 exp
n:iexp

Description

Btst8(m, n) returns -1(true) when the n-th bit in the 64-bit integer expression m is set, and 0 (false) if it's not.

Example

```
Local i As Large = 1000 // 10000 => 1111101000
Print Btst8(i, 8)       // Prints True
```

See Also

Bclr8(), Bset8(), Bchg8()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

<< Operator

Purpose

Shifts a bit pattern left.

Syntax

i = m **<<** n

Description

m<<n shifts the bit pattern of a 32-bit integer expression m, n places left and thereby changes the value in m.

Example

```
Print Bin$(202, 16)       // Prints
  0000000011001010
Print Bin$(202 << 4, 16) // Prints
  0000110010100000
```

Remarks

Shl(m, n) is synonymous with m<<n and can be used instead. As long as the result of the shift does not exceed the given width, m<<n is equivalent to a multiplication of m with 2^n.

See Also

Shl, Shr, Sar, Rol, Ror, >>, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

>> Operator

Purpose

Shifts a bit pattern right.

Syntax

i = m **>>** n

Description

m>>n shifts the bit pattern of a 32-bit integer expressions m, n places right and thereby changes the value in m.

Example

```
OpenW # 1
Print Bin$(202, 16)       // Prints
  0000000011001010
Print Bin$(202 >> 4, 16) // Prints
  0000000000001100
```

Remarks

Shr(m, n) is synonymous with m>>n and can be used instead. As long as the result of the shift does not exceed the given width, m>>n is equivalent to a division of m by 2^n.

See Also

<<, Shl, Shr, Sar, Rol, Ror, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Shr Function

Purpose

Shifts a bit pattern right. **Shr** can be used as a function, as an operator, and as an assignment operator.

Syntax

Shr(m, n)

Shr%(m, n)

Shr&(m, n)n

Shr|(m, n)

Shr8(m, n)

m **Shr** n

m **Shr8** n

Shr v, n

m, n:integer expression
v:ivar

Description

Shr(m, n) and **Shr%** shifts the bit pattern of a 32-bit integer expressions m, n places right (Shr = SHift Right) and, optionally, stores the new value in a variable. **Shr&**(m, n) and **Shr|**(m, n) shift the bit pattern of a 16-bit or an 8-

bit integer expression m respectively, n places right. **Shr8** is used to shift a **Large** integer.

The operators **Shr** and **Shr8** perform a right shift on an integer and Large, respectively.

Shr v, n assignment shifts the value in v by n and returns the value in v. The type of the operation is determined by the type of variable v.

Example

```
Debug.Show
Dim l|, l%
Trace Bin$(202, 16)                 // Prints
  0000000011001010
Trace Bin$(Shr(202, 4), 16)         // Prints
  0000000000001100
l% = Shr(202, 4)
Trace Bin$(Shr%(202, 4), 16)        // Prints
  0000000000001100
l% = Shr%(202, 4)
Trace Bin$(Shr|(202, 4), 8)         // Prints 00001100
l| = Shr(202, 4)
Trace l|                            // Prints 12
```

Remarks

m **>>** n is synonymous with **Shr**(m, n) and can be used instead. As long as the result of the shift does not exceed the given width, **Shr**(m, n) is equivalent to a division of m by 2^n.

x = 100 : 100 **Shr** 3 or **Shr**(100, 3)

100 in binary: 0000 0000 0000 0000 0000 0000 0110 0100
Shift: 0000 0000 0000 0000 0000 0000 0011 0010

Shift: 0000 0000 0000 0000 0000 0000 0001 1001
Shift: 0000 0000 0000 0000 0000 0000 0000 1100

Result is 12 = **CInt**(100 / 8) = **CInt**(100 / 2^3)

x = -8 : -8 **Shr** 4 or **Shr**(-8, 4)

-8 in binary: 1111 1111 1111 1111 1111 1111 1111 0111
Shift: 0111 1111 1111 1111 1111 1111 1111 1011
Shift: 0011 1111 1111 1111 1111 1111 1111 1101
Shift: 0001 1111 1111 1111 1111 1111 1111 1110
Shift: 0000 1111 1111 1111 1111 1111 1111 1111

Result is 258435455.

See Also

Shl, Rol, Ror, <<, >>

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Sar Command

Purpose

Shifts a bit pattern to the right.

Syntax

Sar m, n

m:integer variable
n:iexp

Description

Sar m, n shifts a bit pattern of an integer variable m n steps to the right (Sar = Shift Right), in which the highest bit is copied (and not replaced with zero like with Shr). Each bit shift right is a division by two.

The shift operation to be performed is determined from the data type of the variable.

Example

```
Dim m% = 8, m8 As Large = 8
Sar m%, 2
Print m%          // = 2
Sar m8, 2
Print m8          // = 2
```

Remarks

See **Sar**() for a demonstration of how the bits are shifted.

See Also

Sar(), Shr, Shl, Ror, Rol

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Ror Function

Purpose

Rotates a bit pattern right.

Syntax

Functions:**Ror**(m, n)

Ror|(m, n)

Ror&(m, n)

Ror%(m, n)

Ror8(m, n)

Operators:m **Ror** n

m **Ror8** n

Assignment:**Ror**, ivar, n

m, n:integer expression
ivar:integer variable

Description

Ror(m, n) and **Ror%** shifts the bit pattern of a 32-bit integer expressions m, n places right (Ror = ROtate Right) and "wraps around" the bits moved off the right end to the left end again. The resulting new value is, optionally, stored in a variable. **Ror&**(m, n) and **Ror|**(m, n) rotate the bit pattern of a 16-bit or an 8-bit integer expression m respectively, n places right. **Ror8** rotates a Large integer.

Ror and **Ror8** can be used as operators as well.

Ror ivar, n rotates the value in ivar *n* places and stores the value back in *ivar*.

Example

```
Debug.Show
Local a%, l%, l&, l|
Trace Bin$(202, 32)
// prints 00000000000000000000000011001010
Trace Bin$(Ror(202, 4), 32)
// prints 10100000000000000000000000001100
l% = Ror(202, 4)
Trace l%
// prints -1610612724
Trace Bin$(202, 16)
// prints 0000000011001010
Trace Bin$(Ror&(202, 4), 16)
// prints 1010000000001100
l& = Ror&(202, 4)
Trace l&
// prints -24564
//
Trace Bin$(202, 8)
// prints 11001010
Trace Bin$(Ror|(202, 4), 16)
// prints 10101100
l| = Ror|(202, 4)
Trace l|// prints 172
```

See Also

Sar, Shl, Shr, Rol

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

_Swab Command

Purpose

Exchanges pairs of bytes

Syntax

_Swab src, dest, count

Src, dest, count: ivar

Description

The **_Swab** function copies *n* byte from src, swaps each pair of adjacent bytes, and stores the result at *dest*. The integer *n* should be an even number to allow for swapping. **_Swab** is typically used to prepare binary data for transfer to a machine that uses a different byte order.

Example

```
OpenW 1
Local a$
a = "AbCdEfGhIjKlMnOpQrStUvWxYz"
_Swab V:a, V:a, Len(a)
Print a  //Result: bAdCfEhGjIlKnMpOrQtSvUxWzY
Do : Sleep : Until Me Is Nothing
```

Remarks

Use **Mirror** to swap at the bit-level.

See Also

_Swab8, _SwabL, Mirror

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

_Swab8 Command

Purpose

Exchanges pairs of 8 adjacent bytes

Syntax

_Swab8 src, dest, count

src, dest, count: ivar

Description

The **_Swab8** function copies *n* bytes from *src*, swaps each pair of 8 adjacent bytes, and stores the result at *dest*. The integer *n* should be a multiple of 8 to allow for swapping. **_Swab8** is typically used to prepare binary data for transfer to a machine that uses a different byte order.

Example

```
OpenW 1
Local a$
a = "AbCdEfGhIjKlMnOpQrStUvWx"
_Swab8 V:a, V:a, Len(a)
Print a     // Result: hGfEdCbApOnMlKjIxWvUtSrQ
Do : Sleep : Until Me Is Nothing
```

Remarks

Use **Mirror8** to swap at the bit-level.

See Also

_Swab, _SwabL, Mirror8

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

_SwabL Command

Purpose

Exchanges pairs of 4 adjacent bytes

Syntax

_SwabL src, dest, count

src, dest, count: ivar

Description

The **_SwabL** function copies *n* bytes from *src*, swaps each pair of 4 adjacent bytes, and stores the result at *dest*. The integer *n* should be a multiple of 4 to allow for swapping. **_SwabL** is typically used to prepare binary data for transfer to a machine that uses a different byte order.

Example

```
OpenW 1
Local a$
a = "AbCdEfGhIjKlMnOpQrStUvWxYz12"
_SwabL V:a, V:a, Len(a)
Print a    //Result: dCbAhGfElKjIpOnMtSrQxWvU21zY
Do : Sleep : Until Me Is Nothing
```

Remarks

Use **Mirror** to swap at the bit-level.

See Also

_ Swab, _ Swab8, Mirror

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

HiByte, LoByte Functions

Purpose

Returns the high or low byte of an expression.

Syntax

a| = **HiByte**(x%)

a| = **LoByte**(x%)

a|:8 bits integer

x%:16/32/64 bits integer expression

Example

```
Debug.Show
Local a As Short
a = MakeWord(15, 155)
Trace HiByte(a)         // to read bit 8-15
Trace GetGValue(a)      // the same with GetGValue
Trace GetByte2(a)       // the same with GetByte2
Trace TypeName(HiByte(a))
Debug.Print
Trace LoByte(a)         // to read bit 8-15
Trace GetRValue(a)      // the same with GetGValue
Trace GetByte3(a)       // the same with GetByte2
Trace TypeName(LoByte(a))
```

Remarks

a| = **HiByte**(x&) is identical to a| = **GetGValue**(x%) and a| = **GetByte2**(x%), while a| = **LoByte**(x&) is identical to a| = **GetRValue**(x%) and a| = **GetByte3**(x%)

a| = **HiByte**(x%) is identical to a| = **Byte**(**Shr**(x%,8)) and a| = **LoByte**(x%) is the same as a| = **Byte**(x%), but the second expressions are compatible with the MS-DOS version of GFA-BASIC.

See Also

GetByte2, HiCard(), HiWord(), HiLarge(), LoCard(), LoWord(), LoLarge()

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

HiCard, LoCard Functions

Purpose

Returns the high or low word in a Long expression. The expression is treated as unsigned.

Syntax

a% = **HiCard**(x%)

a% = **LoCard**(x%)

a%:16 bits unsigned integer
x%:32 bits integer expression

Description

HiCard returns the higher 16 bits and **LoCard** returns the lower 16 bits of a value as an unsigned 16-bit expression.

Example

```
Debug.Show
Local a As Int32 = MakeLong(12345, 678)
Trace HiCard(a)
Trace TypeName(HiCard(a))
Trace LoCard(a)
Trace TypeName(LoCard(a))
Trace Card(a)
```

The following example is written as a 16 bit GFA-BASIC for Windows program, only the declaration Local a| is new in. It opens a window and returns the scan code of a pressed function key.

```
OpenW # 1
Dim a|
Do
  GetEvent
  If MENU(11) = WM_KEYDOWN
    a| = LoByte(HiCard(MENU(13)))
    Text 0, 16, Str$(a|) + Space$(1000)
  EndIf
Until MENU(1) = 4
CloseW # 1
```

Remarks

a% = **HiCard**(x%) is identical to a% = **Card**(**Shr**(x%,16)) and a% = **LoCard**(x%) is the same as a% = **Card**(x%), but the second expressions are compatible with the MS-DOS version of GFA-BASIC.

See Also

HiByte(), HiWord(), HiLarge(), LoByte(), LoCard(), LoWord(), LoLarge()

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

HiWord, LoWord Functions

Purpose

Returns the high or low word in a Long expression.

Syntax

a& = **HiWord**(x%)

a& = **LoWord**(x%)

a&:16 bits integer x%:32 bits integer expression

Description

HiWord() and **LoWord**() are used to return the higher and lower 16 bits of a value.

Example

```
Debug.Show
Dim a% = MakeLong(1234, 4321)
Trace HiWord(a%)
Trace TypeName(HiWord(a%))
Trace LoWord(a%)
Trace TypeName(LoWord(a%))
Trace Word(a%)
```

Remarks

a% = **HiWord**(x%) is identical to a% = **Word**(**Shr**(x%,16)), but the second expression is compatible with the MS-DOS version of GFA-BASIC.

LoWord Word SWord Short: These functions are the same and load the value into the eax register and performs a CDWE assembler instruction to extend the lower 16 bits to the upper 16 bits.

See Also

HiByte(), HiCard(), HiLarge(), LoByte(), LoCard(), LoLarge()

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

HiLarge, LoLarge Functions

Purpose

Returns the high and low long word in a Large expression.

Syntax

a% = **HiLarge**(x)

a% = **LoLarge**(x)

a%:32 bits integer
x:64 bits integer expression

Example

```
Debug.Show
Local a As Large = MakeLarge(123456789, 987654321)
Trace HiLarge(a)
Trace LoLarge(a)
```

Remarks

a% = **HiLarge**(x) is identical to a% = **Shr8**(x,32)

See Also

HiByte(), HiCard(), HiWord(), LoByte(), LoCard(), LoWord()

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

Card, UShort and UWord Functions

Purpose

Performs an **And** 65535

Syntax

card = **Card**(m%)

card = **UShort**(m%)

card = **UWord**(m%)

card	*: card expression*
m%	*: 32-bit integer expression*

Description

All three functions perform the same task, namely limiting a 32-bit integer to 16 bits by clearing the bits 16 to 31.

Example

```
OpenW 1 : Win_1.FontName = "Courier"
Dim a% = 100000, m% = 67631, s% = 32
Print Bin$(m%, s%)          //
  00000000000000010000100000110000
Print Bin$(Card(m%), s%)    //
  00000000000000000000100000110000
Print String$(s%, "-")      // -------------------
  -------------
```

```
Print Bin$(a%, s%)            //
 00000000000000011000011010100000
Print Bin$(UShort(a), s%)     //
 00000000000000001000011010100000
Print Bin$(a And 65535, s%) //
 00000000000000001000011010100000
Print UWord(a)                // 34464
```

Remarks

LoCard() is synonymous with these three functions and can be used instead.

See Also

LoCard, Byte(), Word(), Short()

{Created by Sjouke Hamstra; Last updated: 04/03/2017 by James Gaite}

Short & Word Functions

Purpose

Sign extension

Syntax

%= **Short**(m&)

%= **Word**(m&)

%	*: 32-bit long word expression*
m&	*: 16-bit word expression*

Description

Extends a **Word** to a **Long** word. An And 65535 ($0000FFFF) is performed first on the Word. If the result is greater than 32767 (bit 15 is set), 65535 is subtracted from it. This is equivalent to copying the value of bit 15 to bits 16 to 31.

Example

```
AutoRedraw = 1
FontName = "Courier"
Local s% = 32, m% = 2096, a As Double = 100000
Print Bin(m%, s%)                              //
  00000000000000000000100000110000
Print Bin(Word(m%), s%)                        //
  00000000000000000000100000110000
Print String(s%, "-")                          // -------
  -------------------------
```

```
Print Bin(a, s%)                                //
  00000000000000001100001101010000
Print Bin(Word(a), s%)                          //
  11111111111111111000011010100000
Print Bin((a And 65535) - 65536, s%)            //
  11111111111111111000011010100000
Print Bin((a Mod 65536) - 65536, s%)            //
  11111111111111111000011010100000
Print Word(a)                                   // -31072
```

Remarks

These functions load the value into the eax register and performs a CDWE assembler instruction to extend the lower 16 bits to the upper 16 bits.

See Also

Byte(), Card(), SWord, UShort(), UWord()

{Created by Sjouke Hamstra; Last updated: 04/03/2017 by James Gaite}

MakeL2H, MakeL2L Functions

Action

Makes a Long from 2 bytes

Syntax

z% = **MakeL2H**(byte1, byte0)

z% = **MakeL2L**(byte0, byte1)

byte0, byte1:Byte
z%:Integer

Description

MakeL2H() and **MakeL2L** create a 32-bit integer value form two bytes.

Example

```
OpenW 1
Local p%, x%, y%, z%
x% = 10, y% = 20
z% = MakeL2H(x%, y%) // => $0A14
p% = MakeL2L(x%, y%) // => $140A
Print Hex(z%, 4), Hex(p%, 4)
```

See Also

MakeL3H(), MakeL3L(), MakeL4H(), MakeL4L(), MakeLarge(), MakeLargeHiLo(), MakeLargeLoHi(), MakeLong(), MakeLongHiLo(), MakeLongLoHi(),

MakeWord(), MakeWordHiLo(), MakeWordLoHi(), MakeWParam()

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

MakeL3H, MakeL3L Functions

Action

Makes a Long from 3 bytest

Syntax

z% = **MakeL3H**(byte2, byte1, lo)

z% = **MakeL3L**(lo, byte1, byte2)

lo:Byte
byte1, byte2:Byte
z%:Integer

Description

MakeL3H and **MakeL3L** create a 32-bit integer value form three bytes. The high order byte of the long integer is 0.

Example

```
OpenW 1
Local p%, w%, x%, y%, z%
x% = 10, y% = 20, w% = 150
z% = MakeL3H(w%, x%, y%) // => $00960A14
p% = MakeL3L(w%, x%, y%) // => $00140A96
Print Hex(z%, 8), Hex(p%, 8)
```

Remarks

MakeL3L() is the same as RGB().

See Also

MakeL2L(), MakeL2H(), MakeL4H(), MakeL4L(), MakeLarge(), MakeLargeHiLo(), MakeLargeLoHi(), MakeLong(), MakeLongHiLo(), MakeLongLoHi(), MakeWord(), MakeWordHiLo(), MakeWordLoHi(), MakeWParam()

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

MakeL4H, MakeL4L Functions

Action

Makes a Long from 4 bytes

Syntax

z% = **MakeL4H**(hi, byte2, byte1, lo)

z% = **MakeL4L**(lo, byte1, byte2, hi)

hi, lo:Byte
byte1, byte:Byte
z%:Integer

Description

MakeL4H() creates a 32-bit integer value form four bytes. The first byte is placed in the high order byte of the long integer.

MakeL4L() creates a 32-bit integer value form four bytes. The first byte is placed in the low order byte of the long integer.

Example

```
OpenW 1
Local p%, w%, x%, y%, z%
x% = 10, y% = 20, w% = 150
z% = MakeL4H(1, 2, 3, 4) // 01020304
p% = MakeL4L(1, 2, 3, 4) // 04030201
Print Hex(z%, 8)
Print Hex(p%, 8)
```

See Also

MakeL2L(), MakeL2H(), MakeL3H(), MakeL3L(), MakeLarge(), MakeLargeHiLo(), MakeLargeLoHi(), MakeLong(), MakeLongHiLo(), MakeLongLoHi(), MakeWord(), MakeWordHiLo(), MakeWordLoHi(), MakeWParam()

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

MakeLarge Functions

Action

Makes a Large from two 32-bit integers

Syntax

z% = **MakeLarge**(hi, lo)

z% = **MakeLargeHiLo**(hi, lo)

z% = **MakeLargeLoHi**(lo, hi)

hi, lo:Short
z%:Integer

Description

MakeLarge and **MakeLargeHiLo**() create a 64-bit integer value form two 32-bit integers. The first value is placed in the high order longword of the large integer.

MakeLargeLoHi() creates a 64-bit integer value form two 32-bit integers. The first value is placed in the low order longword of the large integer.

Example

```
OpenW 1
Print Hex(MakeLarge(1, 2))       // $100000002
Print Hex(MakeLargeHiLo(1, 2))   // $100000002
Print Hex(MakeLargeLoHi(1, 2))   // $200000001
```

Remarks

See Also

MakeL2L(), MakeL2H(), MakeL3H(), MakeL3L(), MakeL4H(), MakeL4L(), MakeLong(), MakeLongHiLo(), MakeLongLoHi(), MakeWord(), MakeWordHiLo(), MakeWordLoHi(), MakeWParam()

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

MakeLong Functions

Action

Makes a Long from two 16-bit integers

Syntax

z% = **MakeLong**(hi, lo)

z% = **MakeLongHiLo**(hi, lo)

z% = **MakeLongLoHi**(lo, hi)

hi, lo:Short
z%:Integer

Description

Make Long and **MakeLongHiLo**() create a 32-bit integer value form two unsigned 16-bit integers. The first value is placed in the high order word of the long integer.

MakeLongLoHi() creates a 32-bit integer value form two unsigned 16-bit integers. The first value is placed in the low order word of the long integer.

Example

```
Debug.Show
Trace Hex(MakeLong(1, 2))     // $10002
Trace Hex(MakeLongHiLo(1, 2)) // $10002
Trace Hex(MakeLongLoHi(1, 2)) // $20001
```

See Also

MakeL2L(), MakeL2H(), MakeL3H(), MakeL3L(), MakeL4H(), MakeL4L(), MakeLarge(), MakeLargeHiLo(), MakeLargeLoHi(), MakeWord(), MakeWordHiLo(), MakeWordLoHi(), MakeWParam()

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

MakeWParam Function

Action

Makes a 32-bit value from two 16-bit values.

Syntax

z% = **MakeWParam**(lo, hi)

hi, lo:Short
z%:Integer

Description

MakeWParam() creates a 32-bit integer value form two 16-bit integers. The first value is placed in the low order word of the long integer.

Example

```
Debug.Show
Trace Hex(MakeWParam(1, 2), 8)     // 20001
Trace Hex(MakeLongLoHi(1, 2), 8)   // 20001
```

Remarks

This command is the same as the C macros MAKEWPARAM and MAKELPARAM.

MakeWParam is not the same as **MakeLong**.

See Also

MakeL2L(), MakeL2H(), MakeL3H(), MakeL3L(), MakeL4H(), MakeL4L(), MakeLarge(), MakeLargeHiLo(), MakeLargeLoHi(), MakeLong(), MakeLongHiLo(), MakeLongLoHi(), MakeWord(), MakeWordHiLo(), MakeWordLoHi(), MakeWParam()

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

htonl Function

Purpose

The **htonl** function returns the value in TCP/IP network byte order.

Syntax

Card = **htonl** (host-long)

host-long: A 32-bit number in host byte order.

Description

The Windows Sockets **htonl** function converts a unsigned long from host to TCP/IP network byte order (which is big-endian).

The **htonl** function takes a 32-bit number in host byte order and returns a 32-bit number in network byte order used in TCP/IP networks.

See Also

htonl(), htons(), ntohl(), ntohs()

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

htons Function

Purpose

The **htons** function returns the value in TCP/IP network byte order.

Syntax

Card = **htons** (host-short)

host-short: A 16-bit number in host byte order.

Description

The Windows Sockets **htons** function converts a unsigned short (**Card**) from host to TCP/IP network byte order (which is big-endian).

The **htons** function takes a 16-bit number in host byte order and returns a 16-bit number in network byte order used in TCP/IP networks.

See Also

htonl(), htons(), ntohl(), ntohs()

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

ntohl Function

Purpose

The **ntohl** function converts a unsigned **long** from TCP/IP network order to host byte order (which is big-endian).

Syntax

long = **ntohl** (long)

long: A 32-bit number in *TCP/IP network order*.

Description

The Windows Sockets **ntohl** function converts a unsigned long from TCP/IP network order to host byte order (which is big-endian).

The **ntohl** function takes a 32-bit number in TCP/IP network byte order and returns a 32-bit number in host byte order.

Remarks

The **ntohl** function always returns a value in host byte order. If the *netlong* parameter was already in host byte order, then no operation is performed.

See Also

htonl(), htons(), ntohl(), ntohs()

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

ntohs Function

Purpose

The **ntohs** function converts a unsigned short from TCP/IP network order to host byte order (which is big-endian).

Syntax

card = **ntohs** (short)

short: A 16-bit number in *TCP/IP network order*.

Description

The Windows Sockets **ntohs** function converts a unsigned short (Card) from TCP/IP network order to host byte order (which is big-endian).

The **ntohs** function takes a 16-bit number in TCP/IP network byte order and returns a 16-bit number in host byte order.

Remarks

The **ntohs** function always returns a value in host byte order. If the *short* parameter was already in host byte order, then no operation is performed.

See Also

htonl(), htons(), ntohl(), ntohs()

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Add Command, Operator, Function

Purpose

Adds a numeric expression to a numeric variable.

Syntax

Add x, y(assignment command)
% = x **Add** y(operator)
% = **Add**(i, j[,m,...])(function)

x:numeric variable
y:any numeric expression
i, j, m:integer expression

Description

Add x, y adds the expression y to the value in variable x.

The operator i **Add** j and the function **Add**(i, j, …) return the sum of integer expressions. In case one of the parameters isn't an integer, it is converted to a 32-bit value first (using **CInt**).

Example

```
Debug.Show
Dim b# = 1.5
Trace b# Add 3          // CInt(b#) + 3 = 5
Trace Add(b#, 3)        // CInt(b#) + 3 = 5
Add b#, 3 : Trace b#    // b# = 4.5
```

```
b# = 2.5
Trace b# Add 3           // CInt(b#) + 3 = 5
Trace Add(b#, 3)         // CInt(b#) + 3 = 5
Add b#, 3 : Trace b#     // b# = 5.5
```

Remarks

Although the assignment command **Add** can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed.

Instead of **Add** x, y, you can use x = x + y, x := x + y, or x += y. When using integer variables **Add** doesn't test for overflow!

The **Add**(), **Sub**(), **Mul**() and **Div**() functions can be mixed freely with each other. For example

```
l% = Add(5 ^ 3, 4 * 20 - 3)
```

can be written

```
l% = Add(5 ^ 3, Sub(Mul(4, 20), 3))
```

See Also

+, -, *, /F, \, Add, Sub, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

Item, Count, Clear, Remove, Add Methods

Purpose

These methods are provided for each collection. In addition, these methods exist for each Ocx control that contains a collection.

Syntax

object.**Item(** *index* **)**

*object.**Count***

*object.**Clear***

*object.**Remove(*** *index* ***)***

*object.**Add*** *[index], [key], [text], […]*

object:Buttons, ListImages, Panels, ListItems, ColumnHeaders, Nodes, Tabs

object:ToolBar, ImageList, StatusBar, ListView, TreeView, TabStrip

index:Variant

Description

These methods and properties exist for the named collections that are a property of Ocx controls. As a shortcut, these properties and methods exist for the Ocx controls themselves as well. For instance, the **ToolBar** Ocx control contains a **Buttons** collection of **Button** objects. The members in the collection can either be accessed through the **Buttons** collection, but they are also available directly from the **ToolBar** Ocx. To clear the collection you can invoke the **Clear** method from **Buttons**, but also the **Clear** method from **ToolBar**; *ToolBar*.**Buttons**.**Clear** is identical to *ToolBar*.**Clear**.

Item(index)Specifies the position of a member of the collection. If a numeric expression, index must be a number from 1 to the value of the collection's **Count** property. If a string expression, index must correspond to the key argument specified when the member referred to was added to the collection. If the value provided as index doesn’t match any existing member of the collection, an error occurs. **Item** is the default property for a collection. Therefore, the statements are equivalent: MyCollection(1) MyCollection.Item(1)

Count Returns a Long containing the number of items in a collection. Read-only.

ClearRemoves all objects in a collection.

Remove(index)Removes the specified item from a collection. *index* specifies the name or index in the collection of the object to be accessed.

AddAdds a member to a collection object. The syntax for the **Add** method is different for each Ocx collection.

Example

```
Global li As ListItem, n As Int32
OpenW 1
```

```
Ocx ListView lv = "", 10, 10, 200, 300 : .View = 3
  : .GridLines = True : .FullRowSelect = True
lv.ColumnHeaders.Add , , "Column1" :
  lv.ColumnHeaders.Add , , "Column2"
For n = 1 To 20
  lv.ListItems.Add , n , ""  // Can be shortened to
    lv.Add ...
  lv.ListItems.Item(n).AllText = "Item " &
    Format(n, "00") & ";" & Chr(64 + n)  // Can be
    shortened to lv(n).AllText ...
Next n
Ocx Command cmd1 = "Remove Selected Item", 220,
  10, 140, 22
Ocx Command cmd2 = "Remove all Even Items", 220,
  35, 140, 22
Do : Sleep : Until IsNothing(Win_1)

Sub cmd1_Click
  If lv.SelectedCount <> 0 // Make sure an item is
    selected
    Set li = lv.SelectedItem
    lv.ListItems.Remove li.Index
  EndIf
EndSub

Sub cmd2_Click
  Static Int32 cycle = 1
  Select cycle
  Case 1 // Remove Even
    For n = lv.ListItems.Count DownTo 1
      Set li = lv.ListItems(n)  : Debug li.Key
      If Even(Val(li.Key)) = True Then
        lv.ListItems.Remove li.Index
    Next n
    cmd2.Caption = "Delete remaining Items"
  Case 2
    lv.ListItems.Clear
  EndSelect
  Inc cycle
EndSub
```

Remarks

GFA-BASIC 32 supports the following Ocx collections: **Buttons, ListImages, Panels, ColumnHeaders, ListItems, Nodes, and Tabs.**

See Also

Buttons, ListImages, Panels, ColumnHeaders, ListItems, Nodes, Tabs

ToolBar, ImageList, StatusBar, ListView, TreeView, TabStrip

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

Hash Add

Purpose

Adds an element to a hash table.

Syntax

Hash Add name**[**[key$]**]** [**Before** | **After** *idx*] **,** element

Description

Hash Add adds an *element* to the hash table *name*. Optionally, the element can be inserted before or after a specified index *idx*.

Hash Add ht**[** [key$] **] Before** idx, *value*

Hash Add ht**[** [key$] **] After** idx, *value*

An element can also be added without a key. Unless Before or After is used, the element is placed at the end (tail) of the table.

Hash Add ht[], *value* adds a value at the tail of the hash table.

Example

```
Dim ha As Hash Variant, v As Variant
Hash Add ha["new"], 2.3
Hash Add ha[], " a string"
Hash Add ha["Time"] Before 2, Now
Hash Add ha[] After 2, PI
For Each v In ha[]
```

```
  Print ha[$ Each], v
Next
```

Remarks

See Hash for more information on the Hash table.

See Also

Hash Erase, Hash Input, Hash Load, Hash Remove, Hash Save, Hash Sort, Hash Write

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Hash Remove

Purpose

Removes an element from a hash table.

Syntax

Hash Remove name**[** key$ | **%** idx **]**

Description

Hash Remove deletes a single element from a hash table. The element is either indicated by key or by index.

Example

```
Dim ha As Hash Variant
ha["new"] = 2.3
Hash Remove ha["new"]
Print ha[%]
```

or by index:

```
Dim ha As Hash Variant
ha["new"] = 2.3
Hash Remove ha[% 1] // Delete the first element
Print ha[%]
```

...or...

```
Dim ha As Hash Variant
ha["new"] = 2.3
Hash Remove ha[% ha[%]] // Delete the last element
Print ha[%]
```

Remarks

See Hash for more information on the Hash table.

See Also

Hash Add, Hash Erase, Hash Input, Hash Load, Hash Save, Hash Sort, Hash Write

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Hash Erase

Purpose

Deletes the entire hash table.

Syntax

Hash Erase name**[]**

Description

Hash Erase deletes the entire hash table *name* from memory.

Example

```
Dim ha As Hash Variant
ha["new"] = 2.3
Hash Erase ha[]
Print ha[%]      // Prints 0
```

Remarks

See Hash for more information on the Hash table.

See Also

Hash Add, Hash Input, Hash Load, Hash Remove, Hash Save, Hash Sort, Hash Write

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Hash Input, Hash Write

Purpose

Loads a hash table from an ASCII file.

Syntax

Hash Input name**[] ,** file$ | #n

Description

Hash Write saves a hash table in the file *file*$ or in a file with channel #*n*, which is previously opened with **Open**. The hash table is stored in ASCII format and is be reloaded using **Hash Input.**

Example

```
Dim ha As Hash Variant
ha["new"] = 2.3
Hash Write ha[], App.Path & "\hash_ha.dat"
Hash Erase ha[]
Hash Input ha[], App.Path & "\hash_ha.dat"
Print ha["new"]
Kill App.Path & "\hash_ha.dat" // Tidy-up line
```

or

```
Dim ha As Hash Variant
ha["new"] = 2.3
Hash Write ha[], App.Path & "\hash_ha.dat"
Hash Erase ha[]
Open "hash_ha.dat" for Input As # 1
```

```
Hash Input ha[], # 1
Close # 1
Print ha["new"]
Kill App.Path & "\hash_ha.dat" // Tidy-up line
```

Remarks

See Hash for more information on the Hash table.

See Also

Hash Add, Hash Erase, Hash Load, Hash Remove, Hash Save, Hash Sort

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Hash Load, Hash Save

Purpose

Loads or saves a hash table from a file.

Syntax

Hash Load name**[] ,** file$ | #n

Hash Save name**[] ,** file$ | #n

Description

Hash Save saves a hash table in the file *file*$ or in a file with channel #*n*, which is previously opened with **Open**. The hash table is stored in a fast binary format and is reloaded using **Hash Load.**

Example

```
Dim ha As Hash Double // If this is variant, Hash
  Save/Load does not work
ha["new"] = 2.3
Hash Save ha[], App.Path & "\hash_ha.dat"
Hash Erase ha[]
Hash Load ha[], App.Path & "\hash_ha.dat"
Print ha["new"]
Kill App.Path & "\hash_ha.dat" // Tidy-up line
```

or

```
Dim ha As Hash Double // If this is variant, Hash
  Save/Load does not work
ha["new"] = 2.3
```

```
Hash Save ha[], App.Path & "\hash_ha.dat"
Hash Erase ha[]
Open "hash_ha.dat" for Input As # 1
Hash Load ha[], # 1
Close # 1
Print ha["new"]
Kill App.Path & "\hash_ha.dat" // Tidy-up line
```

Remarks

See Hash for more information on the Hash table.

Hash Save/Load does not seem to work with Variants; use Hash Write/Input instead.

See Also

Hash Add, Hash Erase, Hash Input, Hash Remove, Hash Sort, Hash Write

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Hash Sort

Purpose

Sorts a hash table by its keys.

Syntax

Hash Sort name**[] ,** [**Asc | Desc**] [, compmode]

Description

Hash Sort sorts a hash table in ascending **Asc** or descending **Desc** order. The ascending order is the default.

By default, the sort is performed according to the current **Mode Compare** setting. However, it is possible to force the command to sort according to a different mode by specifying the numerical (not string) value of this mode in the *compmode parameter; the possible values for compmode are the same as for Mode Compare.*

Example

```
Dim ha As Hash Variant, v As Variant
ha["new"] = 2.3
ha["Old"] = 2
// ascending
Hash Sort ha[]
Hash_Print("Ascending order")
// descending
Hash Sort ha [] Desc
Hash_Print("Descending order")
// ascending, sorted by using uppercase conversion
```

```
Hash Sort ha [] Asc , -1
Hash_Print("Ascending order using Uppercase
 Conversion")
// or (does the same)
Hash Sort ha [] , -1
Hash_Print("...and the same again")

Sub Hash_Print(t$)
  Print t$ : Print
  For Each v In ha[]
    Print ha[$ Each], v
  Next
  Print
EndSub
```

Remarks

The hash table isn't sorted by the values of the elements like an array, but by its keys!

See Hash for more information on the Hash table.

See Also

Hash Add, Hash Erase, Hash Input, Hash Load, Hash Remove, Hash Save, Hash Write

{Created by Sjouke Hamstra; Last updated: 04/03/2017 by James Gaite}

GoTo Command

Purpose

Unconditional branch

Syntax

GoTo *label*

label:user defined label

Description

Markers are positions within the GFA-BASIC program, used by **Restore** and **GoTo**. Restore mar is always used together with the Data lines. GoTo *label* is an unconditional jump to a previously defined marker *label*.

GoTo can jump either to a label within the main program or within a procedure. A **GoTo** between PROCEDUREs and/or FUNCTIONs is not allowed, and jumps in or out of loops are also forbidden.

Example

```
OpenW # 1
Print "Goto example"
Print
Print "The program is at position 1"
GoTo p2
Print "The program is at position 2"
p2:
Print "The program is at position 3"
```

Remarks

A label might consist of a number (10) or start with alphanumeric character followed by more characters and ended with a semi-colon (p2:).

The label has function scope and cannot be redeclared within the function. However, the same name can be used as a label in different functions.

See Also

Gosub, Exit If

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

On Error Command

Purpose

Turns on the reporting of error messages by the operating system or GFA-BASIC.

Syntax

On Error GoTo label

On Error Resume Next

On Error GoTo 0

label:label

Description

On Error is used to install an error trap in a **Sub**, **Function**, or **Procedure**. The error information can be obtained form the **Err** object. **On Error** is implemented for compatibility reasons with VB, although the **On Error Resume Next** is particular useful to trap errors from OLE (Automation) objects. The preferred way in GFA-BASIC 32 of trapping errors is by using **Try**/**Catch**.

On Error GoTo *label* - Enables the error-handling routine that starts at label specified in the required argument. The label argument is any line label or line number. If a run-time error occurs, control branches to the label, making the error handler active. The specified line must be in the same procedure as the **On Error** statement; otherwise, a compile-time error occurs.

On Error Resume Next - Specifies that when a run-time error occurs, control goes to the statement immediately following the statement where the error occurred where execution continues. Use this form rather than **On Error GoTo** when accessing OLE objects.

On Error GoTo 0 - Disables any enabled error handler in the current procedure.

Example

```
OnErrorStatementDemo()

Sub OnErrorStatementDemo()
  Dim ObjectRef As Object, Msg$
  On Error Resume Next  ' Defer error trapping.
  ' Try to start non existent
  ' object, then test for
  ' Check for likely Automation errors.
  Set ObjectRef = GetObject("MyWord.Basic")
  Trace Hex(Err.HResult)
  If Err.Number = 46
    Msg = "There was an error attempting to open
      the Automation object!" + _
      Err.Description
    MsgBox Msg, , "Deferred Error Test"
  End If
End Sub
```

Remarks

In case of an error **On Error Resume Next** statement continues to execute the next line as if the line is enclosed in a Try: line : Catch : EndCatch block. In fact, GFA-BASIC 32 generates code like this, although optimized, to support the VB error trap mechanism. The generated code is therefore incremented with 8 bytes for each line in code guarded with **On Error Resume Next**. This is true until the trap is disabled using **On Error Goto 0**.

It is advised to use the **Try** / **Catch** method of error trapping as much as possible. The resulting code is smaller and it provides a better overview. In addition, a block guarded with **On Error Resume Next** might easily catch errors originating from a situation that should be handled, not continued.

Additional background information

One of the assembler instructions generated for **Try** and **On Error** is the floating-point command fwait to wait for the FPU to complete the current operation. In case of a floating point error an exception is not generated immediately, but instead deferred to the next floating point operation or a fwait. With slow FPUs, fwait leads to a performance decrease, although, fwait always needs some time to execute. In addition, implicitly GFA-BASIC 32 invokes **Err**.**Clear** with each **Try** and **On Error Resume Next** statement.

See Also

Try, Err

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

On GoSub Command

Purpose

Performs a branch to a local subroutine specified after **GoSub**, depending on the value of the expression after **On**.

Syntax

On n **GoSub** label1, label2, …

n:integer expression
label1,label2, …:label names

Description

A branch to a local subroutine starting with a label is performed, depending on the value of n. If the value of n is less than 1 or greater than the number of labels specified after GoSub, no branch will be invoked. If n is not an integer, a **Trunc**(n) will be performed first and, if needed, a branch will then be taken. After a local subroutine invoked using **On**...**GoSub** is executed the program continues with the first statement after **On**...**GoSub**. If a local subroutine is not invoked, the execution immediately continues with the first statement after **On**...**GoSub**.

No parameters can be passed to a subroutine invoked with **On**...**GoSub**.

Example

```
ColorPrint("Happy Birthday", 1)
ColorPrint("To Me", 2)
```

```
Procedure ColorPrint(a$, opt)
  On opt GoSub pr, pr1
  pr:
  Color 0
  Print a$
Return
  pr1:
  Color 255
  Print a$
Return
EndProc
```

Known Issues

Only one On...Gosub statement can be used in any one procedure, function or sub-routine as that particular program section stops on returning from the statement called. This is shown in this alternative version of the above-listed On-Gosub example below:

```
Tron dbshow // Gives a record of program execution
  in the debug window
ColorPrint("Happy Birthday", 1, "To Me", 2)

Procedure ColorPrint(a$, opt, b$, opt2)
  On opt GoSub pr, pr1
  a$ = b$ : opt = opt2
  On opt GoSub pr, pr1
  pr:
  Color 0
  Print a$
Return
  pr1:
  Color 255
  Print a$
Return
```

```
EndProc

Proc dbshow
  Debug Trace$
EndProc
```

An alternative would be either to embed the **On...Gosub** command within a separate function (as is done in the original example) or to use **Select...EndSelect** (or **Switch...EndSwitch**) instead of the **On** part of the statement as shown below:

```
ColorPrint("Happy Birthday", 1, "To Me", 2)

Procedure ColorPrint(a$, opt, b$, opt2)
  Select opt
  Case 1 : GoSub pr
  Case 2 : GoSub pr1
  EndSelect
  a$ = b$
  Select opt2
  Case 1 : GoSub pr
  Case 2 : GoSub pr1
  EndSelect
Return
  pr:
  Color 0
  Print a$
Return
  pr1:
  Color 255
  Print a$
Return
EndProc
```

See Also

On GoTo, On Call, GoTo, If, Select

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

On GoTo Command

Purpose

Performs a branch to a local label specified after **GoSub**, depending on the value of the expression after **On**.

Syntax

On n **GoTo** label1, label2, ...

n:integer expression
label1,label2, ...:label names

Description

A branch to a local subroutine starting with a label is performed, depending on the value of n. If the value of n is less than 1 or greater than the number of labels specified after GoSub, no branch will be invoked. If n is not an integer, a **Trunc**(n) will be performed first and, if needed, a branch will then be taken. After a local subroutine invoked using **On**...**GoTo** is executed the program continues with the first statement after **On**...**GoTo**. If a local subroutine is not invoked, the execution immediately continues with the first statement after **On**...**GoTo**.

Example

```
OpenW 1
Local a%, n%
n% = 3
On n% GoTo p1, p2, p3, p4, p5, p6
p1:
```

```
Print "Mark p1:"
GoTo p7 :
p2:
Print "Mark p2:"
GoTo p7 :
p3:
Print "Mark p3:"
GoTo p7 :
p4:
Print "Mark p4:"
GoTo p7 :
p5:
Print "Mark p5:"
GoTo p7 :
p6:
Print "Mark p6:"
p7:
End
```

Remarks

See Also

On GoSub, On Call, GoTo, If, Select

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

DoEvents Function

Purpose

Yields execution so that the operating system can process other events.

Syntax

n = **DoEvents**()

n : ivar

Description

DoEvents switches control to the operating-environment kernel. Control returns to your application as soon as all other applications in the environment have had a chance to respond to pending events. This doesn't cause the current application to give up the focus, but it does enable background events to be processed.

This function is extremely useful for programs that are constantly performing operations rather than waiting for user input for refreshing output to a specific window which Windows would otherwise report as being 'Non-Responsive' until the operations stopped for user input and a **Sleep** command was reached.

The **DoEvents** function returns (also when no message is pending) the number of the received messages.

By using **DoEvents** instead of **Sleep**, all simultaneous running programs (also server activities, printer spooler, etc.) will slow down. A loop with **DoEvents** prevents energy saving of a notebook. **DoEvents** was created only to use during long arithmetical calculation operations. The main message loop shouldn't use **DoEvents**, but instead use **Sleep**.

Example

```
OpenW 1
Local n%
n = DoEvents()
Print n // Prints 1
Do : Sleep : Until Me Is Nothing
```

Remarks

The essential difference between **PeekEvent**, which reads only one message a time and **DoEvents**, which handles all pending messages, is that **PeekEvent** stores all messages in the **Menu**() array and **DoEvents** only partially. **Sleep** doesn't use the **Menu**() array at all.

GetEvent and **Sleep** are more alike. Both wait for a message before going on. **Sleep** handles all pending messages, where **GetEvent** only handles one message.

When porting a GFA-BASIC 16 program you shouldn't use **DoEvents** or **Sleep**, but **GetEvent** or **PeekEvent**. By using **GetEvent** or **PeekEvent** you can get problems, if you use Ocx controls in your program.

As a rule: Don’t mix the **Menu**() array handling and Ocx controls. Use **GetEvent**/**PeekEvent** only in programs, that use the **Menu()** array. A program that uses OCXs has to use **Sleep** (and DoEvents).

```
OpenW 1
Do
  Plot Rand(_X), Rand(_Y), Rand(_C)
  DoEvents
  // to see the difference with DoEvents
  // remove the comment before Sleep
  'Sleep
Until Me Is Nothing
```

Never use an empty loop like the following one

```
Do
  DoEvents
Until Me Is Nothing
```

See Also

Sleep, PeekEvent, GetEvent

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

End Command

Purpose

Terminates a GFA-BASIC program.

Syntax

End

Description

End terminates a GFA-BASIC program.

Example

```
Local i%
OpenW # 1 : Win_1.PrintWrap = True
For i% = 1 To 100
  Print i%`
Next i%
End
Print 220
```

Opens a window and prints the digits from 1 to 100. The program then ends. The last line Print 220 is not executed.

```
OpenW 1 , 10, 10, 235, 255
Ocx Command but1 = "click me", 10, 10, 200, 200
Do
  Sleep
Until Me Is Nothing

Sub but1_Click
  CloseW 1
  End
EndSub
```

Known Issues

If **End** is used in the IDE, the IDE can suddenly freeze.

For more information, see here for more details.

See Also

Quit

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

Exit Command

Purpose

Exits a loop.

Syntax

Exit [Do | For]

Description

The **Exit** command makes it possible to exit any loop (For...Next, While...Wend, Repeat...Until and Do...Loop). In contrast to the **GoTo** command, a loop is terminated in an "orderly" fashion by using **Exit**.

In other words, **Exit** always jumps to the first programming statement after the last line of the loop, while GoTo can jump anywhere within a **Procedure** or **Function**.

Exit Do and **Exit For** help to distinguish between the loops and helps in preventing errors.

Example

```
OpenW # 1
Dim e% = 1
Dim i% = 1
Do
  e% *= i%
  Print Str$(i%) + "! = "; Str$(e%, 5)
  If e% > 32000 Then Exit Do
  i% ++
Loop
```

Calculates the factorial and stores the result in the variable e%. The calculation is terminated if the result exceeds 32000.

Remarks

The **If** condition **Then Exit Do** (or Loop) command common to other dialects of BASIC can also be used.

See Also

Goto, Exit If, Exit Sub

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

Stop Command

Purpose

Halts a GFA-BASIC program.

Syntax

Stop

Description

The **Stop** command halts the program on the line with the **Stop** command.

When a program reaches this line GFA-BASIC 32 will show a Message Box with the question: "Really stop?" and you can choose yes or no. This provides the time to select Step mode in the debug tray-icon.

Example

```
Local i%
OpenW # 1, 10, 10, 100, 100, -1
AutoRedraw = 1
For i% = 1 To 100
  If Mod(i%, 10) = 0
    Print i%
    Stop
  EndIf
Next i%
```

Performs a Stop whenever the counter i% is a multiple of 10.

See Also

End

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

PeekEvent Command

Purpose

Monitors menu and window events

Syntax

PeekEvent

Peek_Event

Description

PeekEvent monitors the occurrence of events in menu bars, pop-up menus, and windows. PeekEvent stores the messages read from the message queue in the Menu() array. PeekEvent is not OLE compatible and will not execute event subs.

*The relevant tests must be performed by the programmer. In contrast to **GetEvent**, **PeekEvent** does not wait.*

Example

```
Local i%
OpenW # 1
Dim m$(20)
Data Lissajous , Figure 1 , Figure 2 , Figure 3
Data Figure 4
Data End ,"", Names , Robert , Piere , Gustav
Data Emile , Hugo ,!!
i% = -1
Do
```

```
  i%++
  Read m$(i%)//read in the menu entries
Until m$(i%) = "!!"//marks the end
m$(i%) = ""//terminates a menu
Menu m$()//activates the menu bar
//
Do
  PeekEvent
  If MENU(1) = 1
    Print "A key was pressed"
  Else If MENU(1) = 20
    Print "A menu entry was selected"
  EndIf
Until MouseK = 2 Or Win_1 Is Nothing
CloseW # 1
```

Remarks

PeekEvent is implemented for compatibility with GFA-BASIC 16, but should not be used in OLE programs.

See **DoEvents** for a discussion on **PeekEvent** and **DoEvents**.

See Also

DoEvents, Sleep, GetEvent

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Quit Command

Purpose

Terminates a GFA-BASIC program and returns back to the calling program.

Syntax

Quit [i]

i:integer expression

Description

The **Quit** command terminates the current GFA-BASIC program and returns to the calling program. Optionally, a 16 bit integer value can be returned to the calling program. The following convention applies:

i = 0 the program was executed without error.

i > 0 an internal program error has occurred.

i < 0 an operating system error has occurred.

Example

```
OpenW # 1, 10, 10, 200, 100, -1
Quit
```

Remarks

Compiled programs are terminated with **Quit** as well.

Known Issues

Using **Quit** in the IDE with or without the optional 16-bit integer can lead to an 'Access Violation' error as shown in the example below:

```
Local i As Int16 = 0
OpenW # 1, 10, 10, 200, 100, -1
Try
  Quit i
Catch
  Print Err.Description
EndCatch
```

Of more concern is, if you re-run this program, it will quit, along with the GB32 application running it resulting in any unsaved work being lost.

For more information, see here

See Also

End

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Do...Loop Structure

Purpose

Declares an infinite programming loop.

Syntax

Do

// program segment

Loop

Description

Do...Loop is an endless loop which can only be terminated by the conditional command **Exit If** or unconditional command **Exit Do**.

Example

```
OpenW # 1
Local r
Do
  r = 0
  Input "Enter radius";r
  If r < 0 Then Exit Do
  Print "The circumference of the circle is: "; 2 *
    PI * r
  Print
Loop
Print
Print "End of program!"
```

The program requests the user to enter the radius of a circle. If the entered value is greater than or equal to zero, the circumference of the circle is calculated and displayed.

You are then requested to enter another value. If you enter a negative value the loop is terminated and "End of program!" is displayed.

Remarks

The **Do**...**Loop** statement is the most universal programming loop and it can be used to emulate all other loops:

Example

```
i% = 0                         For i%=1 To n%
Do                             //
                               programsegment
If i% > n% Then Exit Do
// programsegment
Loop                           Next

Do                             While Inkey$ <>
                               "A"
If Inkey$ = "A" Then Exit      //
Do                             programsegment
// programsegment
Loop                           Wend

Do                             Repeat
// programsegment              //
                               programsegment
If Inkey$ = "A" Then Exit      Until Inkey$ = "A"
```

```
 Do
 Loop
```

Even more powerful loop conditions can be created by combining the **Do**...**Loop** with the evaluation part of the **While**...**Wend** and/or **Repeat**...**Until** loops:

```
Local a$ = "ABCDE...Z", b$, n%
Do Until n% > Len(a$)
  Inc n%
  b$ = Mid$(Trim$(a$), n%, 1)
  Print b$;
Loop While Upper$(b$) <> "."
```

Reads a sequential character from string a$, until the end of the string is reached and while the character string starts with something other than a full stop.

```
OpenW 1
Do While MouseK = 0 : Loop
Do While (MouseK And 1)
  Box MouseX, MouseY, Add(MouseX, 10), Add(MouseY,
    10)
Loop Until Upper$(InKey$) = "A"
```

When the left mouse button is pressed it draws a rectangle at the current mouse position, until a lowercase or uppercase "a" is typed on the key-board.

The following loop combinations are possible:

Do ... Loop
Do ... Loop Until
Do ... Loop While
Do ... Wend
Do ... Until

While ... LoopDo While ... Loop
While ... Loop UntilDo While ... Loop Until
While ... Loop WhileDo While ... Loop While
While ... WendDo While ... Wend
While ... UntilDo While ... Until

Repeat ... LoopDo Until ... Loop
Repeat ... Loop UntilDo Until ... Loop Until
Repeat ... Loop WhileDo Until ... Loop While
Repeat ... WendDo Until ... Wend
Repeat ... UntilDo Until ... Until

Do ... Loop
Do ... Loop Until
Do ... Loop While
Do ... Wend
Do ... Until

See Also

For Next, While Wend, Repeat Until

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

For Next Command

Purpose

a programming loop which is executed the specified number of times.

Syntax

For i=x **To** | **UpTo** | **DownTo** y [**Step** z]

// program-segment

[condition] **Exit For**

Exit For If [condition]

Next [i] | **EndFor** [i]

i:avar; any numeric variable

x, y, z:aexp; arithmetic expression

Description

A **For...Next** loop begins by initializing the loop counter i to the specified starting value. With each run the loop counter is incremented or decremented by the specified amount (in case of default by 1, otherwise by the step value in z). When the counter over- or underflows the loop criterion in y, the command after the next Next is unconditionally branched to.

In the following structure:

```
For i = x To y
  // program segment
Next i
```

the loop counter i is incremented by 1 every time the loop runs through **Next** i. The loop ends when i overflows the loop criterion value y.

In the following structure:

```
For i = x To y Step z
  // program segment
Next i
```

Every time the loop runs through **Next** i, the loop counter i is incremented by step amount in z, if this amount is positive, or is decremented by step amount in z, if this amount is negative.

The loop ends when i, for Sgn(z)=1, overflows the loop criterion value y or, for Sgn(z)=-1, underflows the loop criterion value y.

The following structure:

```
For i = x DownTo y
  // program segment
Next i
```

is a special case of:

`For i=x To y Step z`, where z=-1.

The loop counter i is decremented by 1 every time the loop runs through Next i. The loop ends when i underflows the loop criterion value in y. **Step** can also be used with

DownTo to decrement the count by values greater than 1, but it must always have a negative value.

If, at the very beginning of the loop, the loop counter i is already greater than (for **For...To**) or less than (for **For...DownTo** or **For...To...Step** z, when z<0) the loop criterion y, the loop is not executed.

In the following structure:

```
For i = x To i + y
  // program segment
Next i
```

... the `(i + y)` loop criterion is calculated before the loop is started, rather then re-evaluated with every increase in `i`.

Only integers should be used with **For**...**Next** loops NOT decimal/floating point numbers, as with the latter the count may fail to reach the end of the loop - sometimes because the **Single** or **Double** accumulated value is actually larger the the end of loop criterion, sometimes because if a combination of variable types is used, one may not exactly match the others. For more information, see the Remarks section below.

By using an **Exit For** command, the **For**...**Next** loop can be terminated regardless of whether the loop condition is fulfilled.

Finally, **EndFor** can be used in place of **Next**.

Do note, that the loop criterion in the **For**...**Next** loop must always be numeric!

For loop criteria which are not numeric the **While...Wend, Repeat...Until** or **Do...Loop** loops must be used.

Example

```
Local n As Int32, s As Double
// Prints 1 to 7 then exits loop
For n = 1 To 10
  Print n`
  Exit For If n = 7
EndFor n
Print
// Prints 10 down to 1
For n = 10 DownTo 1
  Print n`
Next n
Print
// Print 1.1 to 1.8 then exits loop
For s = 1.1 UpTo 2.2 Step 0.1
  Print s`
  If s NEAR 1.8 Then Exit For
Next s
```

Remarks

As long as different loop counter variables are used the **For**...**Next** loops can be embedded to any number of levels.

As noted above, only Integers should be used in a For...Next loop as, otherwise, the loop may not complete. This, and a workaround, are shown below:

```
looperror(4.2, 4)
newloop(4.2, 4)
Debug.Show

Proc newloop(vm As Double, s As Double)
```

```
  // Alternative by James Gaite 28th March 2018
  Debug "Alternative loop from 0 to" & vm & "
    through" & s & " iterations."
  Local Int32 ct = Round(vm / (vm / s)), v
  For v = 0 To ct : Debug (vm / s) * v : Next v
EndProc

Proc looperror(vm As Double, s As Double)
  // Bug report by Code Labs 28th March 2018
  Debug "BUG double: missing 4.2"
  Local Double v
  For v = 0.0 To CDbl(vm) Step CDbl(vm / s) : Debug
    v : Next v
EndProc
```

See Also

For Each, While Wend, Repeat Until, Do Loop, ExitFor

{Created by Sjouke Hamstra; Last updated: 28/03/2018 by James Gaite}

For Each Command

Purpose

Repeats a group of statements for each element in a collection or hash.

Syntax

For Each *element* **In** *group* [*statements*]
[**Exit For**]
[*statements*]
Next [*element*]

element:variable
group:collection or hash

Description

The **For...Each** block is entered if there is at least one element in *group*. Once the loop has been entered, all the statements in the loop are executed for the first element in *group*. If there are more elements in *group*, the statements in the loop continue to execute for each element. When there are no more elements in *group*, the loop is exited and execution continues with the statement following the **Next** statement.

Any number of **Exit For** statements may be placed anywhere in the loop as an alternative way to exit. **Exit For** is often used after evaluating some condition, for example **If...Then**, and transfers control to the statement immediately following **Next**.

You can nest **For...Each...Next** loops by placing one **For...Each...Next** loop within another. However, each loop *element* must be unique.

Note If you omit *element* in a **Next** statement, execution continues as if *element* is included. If a **Next** statement is encountered before its corresponding **For** statement, an error occurs.

Example

```
Dim Hi As Hash Int, i%
Hash Add Hi["David"], 3
Hash Add Hi["Paul"], 7
Hash Add Hi["Simon"], 5
For Each i In Hi[]
  Print i, Each, Hi[$ Each]
Next
```

or

```
Local f As Form
AutoRedraw = True
OpenW 1 : OpenW 2 : OpenW 3
For Each f In Forms
  Print "Form 1 Name: "; f.Name
Next
Do : Sleep : Until Win_3 Is Nothing
CloseW 2 : CloseW 1
```

Remarks

See Also

For Next, Hash

{Created by Sjouke Hamstra; Last updated: 06/10/2014 by James Gaite}

While...Wend Structure

Purpose

A terminal program loop which runs until the condition at the beginning of the loop is logically "true".

Syntax

While condition
...
// programsegmemt
...
[**Exit Do** | **Exit If**... | **EndDo**]
Wend | **EndWhile**

condition : any numeric, logical or string condition

Description

The start of a **While**...**Wend** loop must contain a numeric, logical or string condition, which is evaluated before each execution of the body of the loop. If the condition is logically "true", the body of the loop is executed. Otherwise, a branch is taken to the program statement immediately after Wend.

The **While**...**Wend** loop is an entry tested loop. This means that the loop executes only when the condition at the beginning of the loop is logically "true". By using an **Exit If...** or **Exit Do** command, the While...Wend loop can be terminated regardless of whether the loop condition is fulfilled. **EndDo** can be used as well.

EndWhile is synoymous with **Wend**

Example

```
While Not Upper$(InKey$) = "A"
  Exit If MouseK = 2
Wend // or EndWhile if you prefer
```

A loop which runs as long as no lowercase or uppercase "a" is entered from the keyboard or the right mouse button is not pressed.

Remarks

The While...Wend loop can be seen as a logical negation of the **Repeat**...**Until** loop, whereby a **While Not** corresponds to an **Until**.

See Also

For, Repeat, Do, For Each

{Created by Sjouke Hamstra; Last updated: 04/03/2017 by James Gaite}

With Command

Purpose

Executes a series of statements on a single object or a user-defined type.

Syntax

With *object* [*statements*]

End With

object: Name of an object or a user-defined type.

Description

The **With** statement allows you to perform a series of statements on a specified object without requalifying the name of the object. For example, to change a number of different properties on a single object, place the property assignment statements within the **With** control structure, referring to the object once instead of referring to it with each property assignment.

With can be used up to 64 levels deep. However, there is no way to access a higher leveled object, unless the object is fully named.

A **With** structure is closed with **End With**.

After executing the **Ocx** or **OcxOcx** command, **With** is implicitly invoked and the properties and methods of the

Ocx are accessible without naming the object. The With is valid until the next **Ocx** or **OcxOcx** command.

Example

```
Ocx Label MyLabel = "", 10, 10
With MyLabel
  .Height = 18
  .Width = 200
  .Caption = "This is MyLabel"
End With
Do : Sleep : Until Me Is Nothing
```

The example illustrates use of the **With** statement to assign values to several properties of the same object.

```
Ocx StatusBar StatusBar1
Dim tmpP As Panels
Set tmpP = StatusBar1.Panels
With StatusBar1
  Print .Width
  With .Panels
    Print .Count
  EndWith
EndWith
Set tmpP = Nothing
Do : Sleep : Until Me Is Nothing
```

Remarks

See Also

Type, Ocx, OcxOcx

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Repeat Until Command

Purpose

A terminal program loop which runs until the condition at the end of the loop is logically "true".

Syntax

Repeat ...
// programsegmemt
...
[**EndDo** | **Exit Do** | **Exit If**...]
Until condition

condition : any numeric, logical or string condition

Description

The end of a **Repeat**...**Until** loop must contain a numeric, logical or string condition, which is evaluated after each execution of the body of the loop. If the condition is logically "true", a branch is taken to the program statement immediately after Until. Otherwise, the body of the loop is executed again.

The **Repeat**...**Until** loop is an exit tested loop. This means that the loop executes at least once and the test, whether or not, the condition is fulfilled is first performed at the end of the loop.

By using an **Exit If...** or **Exit Do** command, the **Repeat**...**Until** loop can be terminated regardless of whether the loop condition is fulfilled. **EndDo** can be used as well.

Example

```
Dim a$
OpenW # 1
Repeat
  a$ = Upper$(InKey$)
Until a$ = "A"
CloseW # 1
```

A loop which runs until lowercase or uppercase "a" is entered from the keyboard.

See Also

For Next, While Wend, Do Loop

{Created by Sjouke Hamstra; Last updated: 04/03/2017 by James Gaite}

If...EndIf Command

Purpose

A conditional branch statement allowing for execution of specific program segments only when a condition is logically "true".

See Remarks for comparisons of floating-point values.

Syntax

If condition [**Then**]
// program segment
[**Else If** condition
// program segment]
[**Else**
// program segment]
>**EndIf**

Condition:any numeric, logic or string condition

Description

The **If...EndIf** statement is, in addition to **Select**...**Case**, the most important command for controlling the program flow. The program segment after an **If...EndIf** statement will be executed if, and only if, the condition immediately following the **If** is logically True. Otherwise, the control is passed to an **Else...If** or **Else** within the same **If...EndIf** structure. If there are no **Else...If** or **Else**, a branch is performed to the statement immediately after the next **EndIf**. The following structure is an exclusive structure.

```
Dim a% = 10
If a% <> 0 Then
  Print "a% <> 0"
Else
  Print "a% = 0"
EndIf
```

This means that the test, if the condition a% <> 0 is logically true, will be performed first. If it is, the first program segment is executed and a branch to the statement following the **EndIf** is taken. If the condition is logically false, the program segment after **Else** is executed and a branch to the statement following the **EndIf** is taken. In no case will both program segments be executed.

This can be extended to an array of exclusive tests:

```
If Mod(42, 4) <> 0
  Print "42 is not fully divisible by 4"
Else If Mod(42, 5)      // means: <> 0
  Print "42 is not fully divisible by 5"
Else If Mod(42, 8)
  Print "42 is not fully divisible by 8"
Else If Mod(42, 9)
  Print "42 is not fully divisible by 9"
Else
  Print "42 is fully divisible by 4,5,8 and 9"
EndIf
```

Gives only '42 is not fully divisible by 4', because the very first condition is logically true. The first condition in the condition list is logically true causes the execution of the first program segment, and then a branch to the statement immediately after **EndIf**. This is irrespective of whether only one, several or all conditions in the condition list are logically true.

True and false

To GFA-BASIC 32 any value that is not 0, is true. Likewise, the value 0 represents false. The condition If 1, therefore, will always evaluate to true, and If 0 always evaluate to false. When you want to test if a condition is true, you can simply include the expression:

If a$ [Then]

This expression evaluates to nonzero (true) when **Len**(a$) > 0, that is, when a$ contains any data.

Multiple conditions

If you want to test whether two conditions are true, you can use the logical AND operator **&&**. The condition is evaluated form left to right. To evaluate to true both conditions must be true. When the first condition is false, the second isn't evaluated.

```
If Len(a$) && height => 100
```

This expression evaluates to true when a$ contains data and the height variable is greater or equal to 100.

Note Do not confuse GFA-BASIC 32's logical AND operator **&&** with the bitwise AND operator **And** or **%&.** The **&&** operator evaluates two Boolean (true or false) expressions to produce a true or false result. The bitwise **%&** (And) operator, on the other hand, works bits (1's and 0's). Would the **&&** operator be replaced by **And**, then both expression are evaluated to be And-ed. Then the result of the bitwise **And** operation is tested for true or false.

```
Dim a% = 10
```

```
If a% = 0 && ff() Then Print "Only one evaluated"
If a% = 0 And ff() Then Print "Both evaluated"

Function ff() As Int
  Debug "ff"
  Return 1
EndFunction
```

To test whether either of two conditions is true (or if both are true), use GFA-BASIC 32's logical OR operator **||**. The condition is evaluated form left to right. To evaluate to true only one of the conditions must be true. When the first condition is true, the second isn't evaluated.

```
If a% = 0 || ff() Then Print "Both evaluated"
If a% || ff() Then Print "Only one evaluated"
```

The logical OR operator **||** is not the same as the bitwise OR operator **Or**, **|**, or **%|.** Replacing **||** with **Or** would first evaluate both conditions, which are then bitwise Or-ed. Then the result of the bitwise or operation is tested for true or false.

```
If a% Or ff() Then Print "Both evaluated"
```

Remarks

Floating-point consistency when comparing floating-point values.

You might want to select the "Improve Floating-point consistency" checkbox in the Compiler tab of the Properties dialog. This options makes sure that before the CPU processes a floating-point value, the value is (re)read from memory (= variable). This is important, because the CPU works with 80-bit floating point values, where variables hold 64-bit values. Out of efficiency reasons the compiler always tries to use current value in the processor register in the next step as well (speed optimization). The following example demonstrates this. The If condition uses the result of d# = a#/b# that is currently in the CPU, which is a 80-bit value. The comparison with the value in c# is always false, because this is 64-bit floating-point value.

```
Dim a# = 2, b# = 3, c#, d#
c# = a# / b#
d# = a# / b#
If d# = c# Then Print "Eq"
```

It is important to get the correct value in the CPU registers before making a comparison. Checking the "Improve floating-point consistency" box is one option. However this influences all floating-point operations and might decrease efficiency.

Another option is to force a reload of the value from d# in the comparison. This loads a 64-bit value into the register and the comparison with c# will be correctly evaluated. To force a reload the processor must be cleared, which is easily done with ~0.

```
Dim a# = 2, b# = 3, c#, d#
c# = a# / b#
d# = a# / b#
~0                ' clear processor
If d# = c# Then Print "Eq"
```

~0 is translated in the assembler instruction *sub eax, eax*. The value is reloaded from d#.

Known Issues

It is sometimes possible to include the If...Then...Else combination on one line as follows:

```
Local a% = 10, b% = 5
If a% = 9 Then a% = 20, b% = 10 Else a% = 9, b% =
 15
Print a%, b%
```

In this case, Endif is superfluous to requirements and should not be used.

However, with some commands such as Print and when Functions are invloved, this structure throws up an error. Hence...

```
Local a% = 10, b% = 5
If a% = 9 Then Print "TRUE" Else Print "FALSE"
```

...will be reformatted by the IDE and result in an error. To get around this, you can use the ':' separator as follows:

```
Local a% = 10, b% = 5
If a% = 9 Then Print "TRUE" : Else : Print "FALSE"
```

These errors occur as the 'Then' keyword is a late addition to GFA and the IDE seems not to have been fully edited to accomodate it.

See Also

Select...EndSelect, NEAR

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

Select and Switch Commands

Purpose

A conditional command which enables execution of specified program segments depending on an integer expression.

Syntax

Select [**Case**] x
[**Case** value1[,value2,...]]
[statements]
[**Case To** value1]
[**Case** value1 **To** [value2]]
[**Default** | **Otherwise | Case Else**]
[statements]
EndSelect

Switch [**Case**] x
[**Case** value1[,value2,...]]
[statements]
[**Case To** value1]
[**Case** value1 **To** [value2]]
[**Default** | **Otherwise | Case Else**]
[statements]
EndSwitch

x:integer expression or a string - only the first four characters of which are significant.
value1,value2,...an integer or string constant of up to four characters

Description

In all instances below the commands **Switch** and **Select** are interchangeable, as are their end statements **EndSwitch** and **EndSelect**. This is shown best by the following statement:

```
Local Int32 a = 4
Select a
Case <3 : Print "Less than three"
Otherwise : Print "More than three"
EndSwitch
```

Select takes one of the **Case** conditional branches depending on the value of "x". The process begins by selecting and evaluating the first **Case** conditional branch, to test if "x" corresponds to at least one of the values after **Case**. If it does, the program segment following this **Case** is executed and a branch is taken to the program line following the **EndSelect**.

If "x" does not correspond to any values in the first **Case** conditional branch the next **Case** is selected. Every **Case** must be followed by at least one value. When entering a list of values its elements must be separated by commas. Furthermore, GFA-BASIC will also accept a range of values.

Case To *value* corresponds to a range of whole numbers whose elements are less than or equal to *value*.

Case *value* **To** corresponds to the range of whole numbers whose elements are greater than or equal to value.

Case *value1* **To** *value2* corresponds to the range of whole numbers whose elements are greater than or equal to value1 and less than or equal to value2.

If no **Case** conditional statement is satisfied the program segment after the optional **Default** is executed and a branch is taken to the program line following the **EndSelect**; if there is no **Default**, a branch to the program line following the **EndSelect** is taken immediately.

The **Select...Case** conditional statement can therefore assume the following structures:

x = value	Case value
x <= value	Case To value
x => value	Case value **To**
(x => value1) And (x<= value2)	Case value1 **To** value2

Example

```
OpenW 1
PrintScroll = 1
Ocx Timer tmr1
tmr1.Enabled = True
tmr1.Interval = 50
Do
  Sleep
Until Me Is Nothing

Sub tmr1_Timer
  Local a%
  a% = Rand(101)
  Select a%
  Case 1 To 50
    Print "Number between 1 and 50"
  Case 51 To 99
    Print "Number between 51 and 99"
  Case 0, 100
    Print "Number is either 0 ro 100"
  EndSelect
EndSub
```

Remarks

Otherwise or **Case Else** can be used instead of **Default**.

Notice that the **Select Case** structure evaluates an expression once at the top of the structure. In contrast, the If...Then...Else structure can evaluate a different expression for each **ElseIf** statement. You can replace an If...Then...Else structure with a **Select Case** structure only if the **If** statement and each **ElseIf** statement evaluates the same expression. The **Select...Case** if often considerably faster than **If...ElseIf**.

Despite previous documentation stating otherwise, **Select...Case** can be used with strings but only up to a maximum length of four characters. This is because, by default, **Select...Case** assumes an integer result to any evaluation and, if a string is passed, it simply copies in up to the first four characters of that string into the memory area reserved for the integer. This can be best shown in the following example:

```
test("a")
test("AB")

Procedure test(a$)
  // Due to the way Select works, the Case
    statements can either use the string value...
  Select a$
  Case "A" : Print "That was an A (select by string
    value - upper case)"
  Case "AB" : Print "That was AB (select by string
    value - upper case)"
  EndSelect
  // ...or an integer made from the ASCII values of
    the string...
  Select a$
  Case $41 : Print "That was an A (selected by
    numerical value)"
  Case $4241 : Print "That was AB (selected by
    numerical value)"
  EndSelect
  // ...BUT any strings used must be case specific.
  Select a$
  Case "a" : Print "That was an 'a' (select by
    string value - lower case)"
  Case "ab" : Print "That was 'ab' (select by
    string value - lower case)"
  EndSelect
EndProcedure
```

Known Issues

It is possible within the IDE to leave a 'blank' Case section as below:

```
OpenW 1
Local Int32 a = 1
Select a
Case 1  // 'Blank' Case section
Case 2 : Print "Not 1"
Default : Print "Not 1 or 2"
EndSelect
Do : Sleep : Until Win_1 Is Nothing
```

This is useful if no action is to be taken for a certain value or range of values: the above example prints nothing in the IDE. HOWEVER, when the above code is compiled, any blank Case sections are ignored and any value or action contained in the next Case section or, if there are no more, the **Default** or **Otherwise** section is performed instead; hence the above example, if compiled, prints 'Not 1'. This is a known error to which there is a simple workaround: for the Case section that would normally be left blank, add a piece of code that does nothing; e.g. in the above example, rather then leave the case blank, the expression `a = a` can be used.

See Also

If...EndIf

{Created by Sjouke Hamstra; Last updated: 08/03/2018 by James Gaite}

Call command

Purpose

Transfers control to a **Sub, Procedure**, **Function,** or DLL procedure.

Syntax

[Call] subroutine [paramlist]

Description

You are not required to use the **Call** keyword when calling a procedure. The parameters in the *paramlist* may be enclosed in parentheses.

Example

```
Global a$
a$ = "GFA"
Call test_it(a$)
Do
  Sleep
Until Me Is Nothing

Sub test_it(a$)
  OpenW 1
  Text 10, 10, "Hallo " + a$
EndSub
```

See Also

@

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

C:()(), CCall() Function

Purpose

Executes a subroutine at a specified address and returns a 32-bit integer value.

Syntax

x = **C:(**addr**)([**parameters]**)**

x = **CCall(**addr**)([**parameters]**)**

x, addr:iexp
parameters:aexp

Description

The **C:()()** and **CCall()()** functions call a C or an assembler subroutine at address addr%. The parameters are placed in from right to left on the stack. The last parameter is the first on the stack.

C:()() returns with a simple *ret* instruction. The caller must correct the stack.

The parameters can be coerced to a specific format by preceding the value with one of the following designators:

Dbl: double
Sng: float, single
Large: Large integer
Cur: Currency value
L: Long

Int: Integer
Var: Variant

Example

Remarks

The stack:

a% = **CCall**(addr%)(1, 2, 3) or a% = **C:**(addr%)(1, 2, 3)

12[esp]3

8[esp]2

4[esp]1

[esp]Return address

The routine that is called doesn't correct the stack pointer.

See Also

LC:(), P:(), LP:(), Call(), CallX(), LCCall(), PasCall(), LPasCall(), StdCall(), LStdCall()

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

LC:()(), LCCall()() Function

Purpose

Executes a subroutine at a specified address and returns a 64-bit integer (Large) value.

Syntax

x = **LC:(**addr**)([**parameters]**)**

x = **LCCall(**addr**)([**parameters]**)**

x:64-bit integer
addr:iexp
parameters:aexp

Description

The **LC:()()** and **LCCall()()** functions call a C or an assembler subroutine at address addr%. The parameters are placed in from right to left on the stack. The last parameter is the first on the stack.

LC:()() return with a simple ret instruction. The caller must correct the stack.

The parameters can be coerced to a specific format by preceding the value with one of the following designators:

Dbl: double

Sng: float, single

Large: Large integer

Cur: Currency value

L: Long

Int: Integer

Var: Variant

Example

Remarks

See Also

C:(), P:(), LP:(), Call(), CallX(), CCall(), PasCall(), LPasCall(), StdCall(), LStdCall()

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

LP:()(), LPasCall() Function

Purpose

Executes a subroutine at a specified address and returns a Large value.

Syntax

x = **LP:(**addr**)([**parameters]**)**

x = **LPasCall(**addr**)([**parameters]**)**

x:Large
addr:iexp
parameters:aexp

Description

The parameters are placed in reverse order on the stack.

LP:()() and **LPasCall()()** expects the subroutine to clear the stack.

The parameters can be coerced to a specific format using by preceding the value with one of the following designators:

Dbl: double

Sng: float, single

Large: Large integer

Cur: Currency value

L: Long

Int: Integer

Var: Variant

Example

```
Dim a% = ProcAddr(test)
~LP:(a%)( Large:2, 3 )
' or
~LPasCall(a%)( Large:2, 3)

Procedure test(i%, la As Large)
  Print la, i%
EndProc
```

Remarks

A Procedure takes it parameters by value using the StdCall convention.

See Also

C:(), LC:(), P:(), LP:(), Call(), CallX(), CCall(), LCCall(), PasCall(), LPasCall(), StdCall(), LStdCall()

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Call()(),CallX()() Command

Purpose

Executes a subroutine at a specified address.

Syntax

Call(addr**)([**parameters]**)**

CallX(addr**)([**parameters]**)**

x, addr:iexp
parameters:registers

Description

The **Call()()** and **CallX()()** functions call a C or an assembler subroutine at address addr%. Any arguments are passed through the pseudo registers _EAX, _ECX, etc. **CallX** allows passing segment registers - _DS, _ES, _FS, and _GS - as well.

Example

```
Debug.Show
Dim a$ = Space$(30)
Call (LabelAddr(xMemClr)) ( _EDI = V:a$, _ECX =
  30)
Trace a$
If 0
  xMemClr:   . mov al, 67 :   . rep stosb : . ret
EndIf
```

Remarks

Call()() and **CallX()()** don't use the stack. A subroutine should return with a simple *ret* instruction.

See Also

LC:(), P:(), LP:(), CCall(), LCCall(), PasCall(), LPasCall(), StdCall(), LStdCall()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

StdCall() Function

Purpose

Executes a subroutine at a specified address and returns a Long value.

Syntax

x = **StdCall(**addr**)([**parameters]**)**

x:Long
addr:iexp
parameters:aexp

Description

StdCall()() expects the subroutine to clear the stack. The parameters (when 4 bytes in size) are placed ont the stack as follows:

a% = StdCall(addr%)(1, 2, 3)

12[esp] 3

8[esp] 2

4[esp] 1

[esp] return address

The called routine should end with 'ret 12' correcting the stack.

The parameters can be coerced to a specific format by preceding the value with one of the following designators:

Dbl: double
Sng: float, single
Large: Large integer
Cur: Currency value
L: Long
Int: Integer
Var: Variant

Example

```
Dim a% = ProcAddr(test)
~StdCall(a%)( Large:2, 3 )

Procedure test(la As Large, i%)
  Print la, i%
EndProc
```

Remarks

A **Procedure** takes it parameters by value using the StdCall convention. StdCall is the default calling convention for GFA-BASIC 32 and Windows.

See Also

C:(), LC:(), P:(), LP:(), Call(), CallX(), CCall(), LCCall(), PasCall(), LPasCall(), StdCall(), LStdCall()

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

LStdCall() Function

Purpose

Executes a subroutine at a specified address and returns a Large value.

Syntax

x = **LStdCall(**addr**)([**parameters]**)**

x:Large
addr:iexp
parameters:aexp

Description

LStdCall()() expects the subroutine to clear the stack.

The parameters can be coerced to a specific format using by preceding the value with one of the following designators:

Dbl: double

Sng: float, single

Large: Large integer

Cur: Currency value

L: Long

Int: Integer

Var: Variant

Example

```
Dim a% = ProcAddr(test)
~LStdCall(a%)( Large:2, 3 )

Procedure test(la As Large, i%)
  Print la, i%
EndProc
```

Remarks

A Procedure takes it parameters by value using the StdCall convention. StdCall is the default calling convention for GFA-BASIC 32 and Windows.

See Also

C:(), LC:(), P:(), LP:(), Call(), CallX(), CCall(), LCCall(), PasCall(), LPasCall(), StdCall(), LStdCall()

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

$AutoPost directive

Purpose

Enables automatic recognition of variable name postfixes.

Syntax

$AutoPost[fix][On|Off]
$NoAutoPost[fix]

Description

$AutoPost enables - and **$NoAutoPost** disables - the automatic recognition of postfixes with variable names.

By default, GFA-BASIC 32 recognizes variables without a postfix after they are declared with a postfix (Local i% : i = 12). This is only possible when the name is used with one type; if the variable name is used for an integer it cannot then be used for a string at the same time.

$AutoPost is synonymous with **$AutoPostOn**, **$AutoPostfix** and **$AutoPostfixOn**

$NoAutoPost is synonymous with **$NoAutoPostfix**, **$AutoPostOff** and **$AutoPostfixOff**

Example

```
Dim a$
$AutoPostOn ' Postfix recognition enabled
a = "GFA"   ' a is recognised as a$
$NoAutoPost ' Disable postfix recognition
```

```
a = "GFA"    ' Does not compile - IDE Error:
  "Variable a?"
```

Remarks

The default setting differs from the default behaviour of GFA-BASIC for Windows 16-bit. In the 16-bit version, a name could be used many times, but each occurence would still be different because of the use of a postfix (Local i%, i$). When a 16-bit program is ported to 32-bit the compiler might be instructed to use the postfix to differentiate between the variables. To make sure that the variables are used with a postfix explicitly use **$NoAutoPost**.

In addition, **$NoAutoPost** only works when variables are declared with a postfix; if a variable is declared using the **Dim** *variable* **As** *vartype* format, it is unaffected by the **$AutoPost** settings and takes precedence over any variables declared using a postfix as shown in the example below:

```
Dim a& = 3, a% = 4, a As Int32
$AutoPost
a = 5                // Assigns value to a not a&
  or a%
$NoAutoPost
a = 7                // Again assigns value to a
  not a& or a%
Print a&, a%, a      // Prints 3    4    7
```

$ObjCheck or **$Obj** re-enables auto post recognition as well.

See Also

$ArrayChk, $For, $Obj, $Step

{Created by Sjouke Hamstra; Last updated: 23/06/2015 by James Gaite}

$ArrayCheck directive

Purpose

A code optimization directive which can be used to switch off or turn back on array boundary checking.

Syntax

$**ArrayCheck**[**On** | **Off**]
$**ArrayChk**[**On** | **Off**]

Description

This directive can be used to temporarily disable the checking in a portion of the code. **$ArrayCheckOff** disables the checking of array boundaries. **$ArrayCheckOn** enables the checking again. The code in between these two directives will not protected against array boundary overflow.

For finished programs, checking of array boundaries with each array access may not be considered necessary and, as it takes additional code steps and requires extra execution time to provide array index checking, it may be beneficial to switch it off. The default setting for array index checking is controlled in the Compiler Properties dialog box but always keep in mind that the ArrayCheck directive overrides this default setting.

Example

```
$ArrayCheckOff    ' Disable boundary checking
Dim arInt%(1)
```

```
' Assign a value to the third element (array
  starts at element 0)
' Since error checking is disabled the program
  doesn't report an error.
arInt%(2) = 1 : Print "No Error"
$ArrayChkOn      ' Enable checking again
' The following code uses array checking once
  again
arInt%(2) = 1    ' Causes an 'Array-Bounds-
  Exceeded' error
```

Remarks

ArrayCheckOn is synonymous with **ArrayChkOn**, as **ArrayCheckOff** is with **ArrayChkOff**.

See Also

$AutoPost, $For, $Obj, $Step

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

$For directive

Purpose

Disables overflow checking for For...Next statements

Syntax

$ForFast
$ForNoOver[flow[Check]]
$ForNoCheck[Over[flow]]

$ForSlow
$ForOver[flow[Check]]
$ForCheck[Over[flow]]

Description

$ForFast, or any one of the **$ForNo..** variants, disables overflow checking of the count variable within a For...Next loop while $**ForSlow**, or any one of the **$For..** variants, enables it again; the default state is enabled.

Overflow checking disabling is only possible with integer count variables. The performance gain is about 30% for an empty loop.

Example

```
Dim a$, i%, t As Double
$ForFast    ' Disable overflow checking
t = Timer
For i% = 0 To 1000000 : a$ = Str(i) : Next i%
Print "ForFast: "; Timer - t
```

```
$ForSlow     ' Enable overflow checking again
t = Timer
For i% = 0 To 1000000 : a$ = Str(i) : Next i%
Print "ForSlow: "; Timer - t
```

Remarks

For...Next loops until **_maxInt** are only possible with overflow check enabled. The following example would normally loop 101 times before the count variable i% will overflow (**_maxInt** to **_minInt**, 2147483647 to -2147483648, 0x7fffffff to 0x80000000). Normally, the loop is ended, however with $ForFast no overflow check is performed and results in an infinite loop.

```
Local i%, j%
j% = _maxInt
$ForFast
For i% = j% - 100 To j%    // Loop using $ForFast
  Print i
Next
```

See Also

$AutoPost, $ArrayChk, For...Next, $Obj, $Step

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

$Obj directive

Purpose

Error checking for OLE-Object types.

Syntax

$ObjNoErr | **$ObjectNoErr**

$ObjCheck | **$ObjectCheck**

Description

$ObjNoErr or **$ObjectNoErr** disables (temporarily) the error checking for OLE object types. Similar to array access, each object access is encapsulated in error checking code and every method call or property access is guarded. This requires some additional code and execution time (the default setting). Without the checking you will save some code (4 bytes) per OLE property or method call and as a result the code will execute faster because no checks are performed.

With **$ObjCheck** the error checking is re-enabled. See HResult for more information.

Example

```
$ObjNoErr
OpenW 1
Ocx CommDlg cd
cd.Flags = cdfScreenFonts | cdfShowHelp
cd.ShowFont
cd.Flags = cdcShowHelp
cd.ShowColor

Sub cd_OnHelp
  Me.Caption = "Help Requested"
EndSub
```

Remarks

Normally object calls don't return error values; however it is advisbale to still use $ObjNoErr with caution.

See Also

HResult, $AutoPost, $ArrayChk, $For, $Step

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

$Step

Purpose

Option to set single step (debug) mode at subroutine level.

Syntax

$Step[On | Off]

Description

$StepOff switches off the insertion of single step code before each code line. The code affected can no longer be debugged using the debug icon in the tray. In addition Ctrl-Break is disabled as well. This setting only affects code running inside the IDE.

$StepOff is used at the procedure level. Once a procedure is fully tested and error free the $StepOff directive speeds up the execution time and reduces the size of the subroutine. It'll save 5 bytes before each code line and reduces the speed about 18 cycles per line.

$StepOn re-enables the insertion of debug code before each line.

$Step (without On or Off) enables a single insertion of debug code, without disturbing the global setting. This could be useful for guarding a loop, so that the program can be stopped using Ctrl-Break.

Example

```
$StepOff
Print Trial(1750000)
$StepOn
Print Trial(1750000)

Function Trial(value%)
  Dim i As Int, t As Double = Timer
  For i = 0 To 2000000
    If i = value% Then Return Timer - t
    $Step
  Next i
  Return Timer - t
EndFunc
```

NOTE: As with any timed example, other background routines may distort the results. In general, the second time value shown should always be higher than the first.

Remarks

When the program is compiled to an executable all $Step code is removed. This directive is of use only in the IDE.

Naked procedures have the **$StepOff** directive by default.

See Also

$AutoPost, $ArrayChk, $For, $Obj, Naked

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

Naked Modifier

Purpose

Faster execution of subroutines.

Syntax

Sub | **Proc**[**edure**] | **Function**[**Var**] *name* () [**As** *Type*] **Naked**

Description

It is important to understand that the GFA-BASIC 32 **Naked** modifier isn't the same as the *naked* keyword in MSVC. In GFA-BASIC 32 **Naked** instructs the compiler to generate a minimum of prologue and epilogue code, in MSVC *naked* doesn't generate prologue and epilogue at all. In fact, the GFA-BASIC 32 **Naked** attribute generates the same prologue and epilogue code MSVC does for a normal function. **Naked** instructs the compiler to generate code much like C. In GFA-BASIC 32 **Naked** results in the fastest possible code (assembler excluded).

Using **Naked** comes with a severe penalty, though. All safety nets are removed and an exception definitely crashes the program. **Try**/**Catch** cannot be applied to **Naked** subroutines, as well as debugging. **Naked** implicitly implies **$StepOff** for the entire subroutine. Local variables that require additional memory of the heap are to be released explicitly. For a local string, variant, and array, the descriptors are placed on the stack and will be removed, but the allocated memory isn't. A string must be released by setting it to "" and an array must be erased (**Erase**). A

Variant must be assigned a safe value (Int, Float, whatever as long as it doesn't require additional memory). Any objects that are referenced must be set to **Nothing** explicitly.

From the above it is clear that a normal subroutine performs quite some housekeeping. The normal prologue code of a GFA-BASIC 32 routine sets up a table for all local variables and releases their contents at the end of the routine (**EndSub**, **Exit Proc**, **Return**, etc). It also includes code to step through the code line by line and keeps record of the current executing line so that in case of an error the line can be marked in the editor. Finally, it includes code to create an error trap using **Try**/**Catch** or **On Error**. Everything that makes BASIC programming easy is left out when **Naked** is applied. **Naked** is for advanced programmers only, although some subroutines might be naked without much background knowledge. See example.

Example

```
Local t1#, t2#, n As Int32, a$ = "A", res?
t1# = Timer : For n = 1 To 100000 : res? =
  IsAlpha(Asc(a$)) : Next n : t1# = Timer - t1#
t2# = Timer : For n = 1 To 100000 : res? =
  IsAlpha_nn(Asc(a$)) : Next n : t2# = Timer - t2#
Print "Time Test for IsAlpha:"
Print "Naked version: "; Format(t1#, "0.######");
  " secs"
Print "Normal version: "; Format(t2#, "0.######");
  " secs"
Print "Performance Increase: "; Format((t2# / t1#)
  - 1, "###%")
Print
t1# = Timer : For n = 1 To 100000 : res? =
  IsAlnum_(Asc(a$)) : Next n : t1# = Timer - t1#
```

```
t2# = Timer : For n = 1 To 100000 : res? =
  IsAlnum_nn(Asc(a$)) : Next n : t2# = Timer - t2#
Print "Time Test for IsAlnum:"
Print "Naked version: "; Format(t1#, "0.######");
  " secs"
Print "Normal version: "; Format(t2#, "0.######");
  " secs"
Print "Performance Increase: "; Format((t2# / t1#)
  - 1, "###%")
Print
t1# = Timer : For n = 1 To 100000 : res? =
  IsUpper(Asc(a$)) : Next n : t1# = Timer - t1#
t2# = Timer : For n = 1 To 100000 : res? =
  IsUpper_nn(Asc(a$)) : Next n : t2# = Timer - t2#
Print "Time Test for IsUpper:"
Print "Naked version: "; Format(t1#, "0.######");
  " secs"
Print "Normal version: "; Format(t2#, "0.######");
  " secs"
Print "Performance Increase: "; Format((t2# / t1#)
  - 1, "###%")

Function IsAlpha(a As Int) As Bool Naked
  // Alphabetic (A - Z or a - z)
  IsAlpha := (a >= 65 && a <= 90) || ( a >= 97 && a
    <= 122)
EndFunction

Function IsAlpha_nn(a As Int) As Bool
  // Alphabetic (A - Z or a - z)
  IsAlpha_nn := (a >= 65 && a <= 90) || ( a >= 97
    && a <= 122)
EndFunction

Function IsAlnum_(a As Int) As Bool Naked
  // Alphanumeric (A - Z, a - z, or 0 - 9)
  IsAlnum_ := (a = 95) || (a >= 48 && a <= 57) _
```

```
    || (a >= 65 && a <= 90) || ( a >= 97 && a <=
     122)
EndFunction

Function IsAlnum_nn(a As Int) As Bool
  // Alphanumeric (A - Z, a - z, or 0 - 9)
  IsAlnum_nn := (a = 95) || (a >= 48 && a <= 57) _
    || (a >= 65 && a <= 90) || ( a >= 97 && a <=
     122)
EndFunction

Function IsUpper(a As Int) As Bool Naked
  IsUpper := (a = Asc(Upper(Chr(a))))
EndFunction

Function IsUpper_nn(a As Int) As Bool
  IsUpper_nn := (a = Asc(Upper(Chr(a))))
EndFunction
```

Remarks

A normal GFA-BASIC 32 subroutine does not guarantee anything about the contents of processor registers when exiting and returning to the caller. Just before returning GFA-BASIC 32 calls a library function that clears the local variables and resets the stack. In the process register variables are used and any value assigned to the register is deleted. This is why a **Procedure** used as a call back function that returns a value through the eax register must be **Naked**; the library call to release the local variables is not made. Therefore, you will see procedures like these:

```
Proc WndProc(hWnd As Handle, msg As Int, wParam As
  Int, lParam As Int) Naked
  Local RetVal
  //... Code ...
  Asm mov eax, [RetVal]
EndProc
```

However, when a **Function** is used as a call back subroutine, you can simply use the **Return** statement to return a value to the caller. Values returned from a **Function** are always passed in the eax register. Now the subroutine doesn't need to be **Naked** and **Try**/**Catch** error trapping can be implemented.

Naked must be used when porting ***_fastcall*** functions. Without **Naked** GFA-BASIC 32 puts prologue and epilogue code in the function that obscures the registers used for parameter passing and returning.

The next sample shows the amount of stack memory for a recursive function. Note that the string is allocated in the caller. A special string optimizing feature of the compiler allows this construction.

```
Print // OpenW 1
Print abc("test", 9)
Do
  Sleep
Loop Until Me Is Nothing

Function abc(a$, c%) As Int Naked
  Local r%
  Static p% = V:r
  Print V:r - p
  If c% > 0
    abc = abc(a$, c% - 1) + 1
  EndIf
EndFunc
```

See Also

Sub, Procedure, Function, $StepOff

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

Auto Command

Purpose

automatic collection and declaration of undeclared variables as global variables

Syntax

Auto

Description

Auto is mainly implemented to convert GFABASIC 16 source codes (LST files) which don't contain explicit declarations of variables. In GFA-BASIC 32 global variables must be declared before they are used. To comfort and collect all undeclared variables **Auto** should be used at the top of the program. **Auto** collects undeclared variables and appends them to the **Auto** code line when Shift+F5 is pressed (test compile).

By replacing **Auto** with **Global** the variables are declared more permanently. Any **Auto** command instructs the compiler to make an extra pass. All variables after **Auto** are deleted and comments and changes will disappear. Then the variables are collected, sorted, and inserted in the code after the **Auto** command. Variables that are followed with a parenthesis are generated as Auto x() As Double, an array without elements.

After collecting the variables they must be carefully examined to make sure their type is correct. String variables may be declared as Variant, and integers as Large, when Long suffices.

Variables without postfix default to Double.

Example

```
Auto
a% = 1
a$(0) = 1
test(b)

Sub test(tst$)
EndSub
```

becomes after Shift+F5:

```
Auto a$(), a%, b As Double
a% = 1
a$(0) = 1
test(b)

Sub test(tst$)
EndSub
```

Note that b has gotten a wrong type!

Also note that the a$() array is undefined and will result in an 'Array Bounds Exceeded' error.

Remarks

In 16 Bit GFA-BASIC it was allowed to use the same variable name for different types: a, a%, a$, a%() and a(). In GFA-BASIC 32 variables and function names must be different, as well as simple variables and array names.

VB has a greater limitation, each name must be unique. In GFA-BASIC 32 a$ is different than a#, but in VB this isn't allowed.

See Also

Sub, Procedure, Function, Global, Dim

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

IsExe Function

Purpose

Returns a Boolean value indicating a whether a programming is running as a standalone EXE or inside the IDE.

Syntax

Bool = **IsExe**

Description

Example

```
MsgBox0 "I'm running" & Iif(IsExe, " as a stand-
  alone EXE!", " inside the IDE.")
```

Remarks

See Also

App

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Monitor Command

Purpose

Sets a debugger breakpoint for an external debugger.

Syntax

Monitor[n]

n:integer expression

Description

Monitor [n] calls interrupt $3 and passes the value n in processor registers AX and DX. The command is intended for inserting of breakpoints in compiled programs.

See Also

-

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

CallTree Function

Purpose

Returns a string containing called subroutines.

Syntax

$ = CallTree [(start% [,end%])]

Description

CallTree returns a string with a list of procedure calls up to the position CallTree is invoked, this is called procedure call tree hierarchy.

The number of entries that **CallTree** must return can be limited by using the optional parameters start% and end%. If start% < end% one entry is returned: entry start%. **CallTree**(1, 3) will return the first three entries of the call tree hierarchy (list). **CallTree**(3, 3) or **CallTree**(3, 0) return entry 3 of the list. If start% <= 0, then end% will be ignored. Slicing a part of the list is mainly interesting within recursively called functions.

CallTree(-1) returns the approximate number of entries in the list

CallTree(0) same as **CallTree**

CallTree(1) returns the name and the parameter of the actual called Procedure, Sub, or Function.

CallTree(2) returns the name and parameters that calls the actual Procedure, Sub, or Function.

CallTree(3) returns the name and the parameters that has called the one returned from **CallTree**(2).

CallTree is to be used for debugging purposes. It is a feature independent of other debugging facilities of GFA-BASIC 32. **CallTree** is used at the start a subroutine body to find out which subroutines called it and which routines called the caller. This list provides a kind of cross reference of calling procedures. The example shows the **CallTree** for the function *rt*() each time it is called.

Example

```
Ocx ToolBar tb1
tb1.AddItem
Me.BackColor = colBtnFace
t(1, 12, tb1.Button(1), , Me)
Do
  Sleep
Loop Until Me Is Nothing

Sub t(a#, b%, c , Optional ox, d)
  Local j% = 9    // dummy
  Print rt(4)     // calculate faculty
EndSub

Function rt(ByVal i|) As Double   // Faculty
  Print CallTree                 // show in Win_1
  Local h As Hash String
  Split h[] = CallTree, "\r\n"
  qq(h[])                        // another way
  MsgBox "levels: " & CallTree(-1) _
    & #13#10 & CallTree(1, 3)     // in a msgbox
  Debug.Print "CallTree"          // in the
    output...
  Debug.Print CallTree            // ...window
  If(i > 1) Then Return rt(i - 1) * i
  Return 1
End Func

Sub qq(hs As Hash String)
  Local a$
  For Each a In hs[]
    ' Print a // Copies CallTree output to screen
  Next
EndSub
```

This program calculates the faculty of a value by recursively calling *rt()*. The Function *rt()* shows 4 possible ways of inspecting the call tree hierarchy. First it prints the call list in the client area of the window. Then the list is split in to a Hash array and then the Hash is 'printed' into the client area as well. Third, the list is displayed using a Message Box. Finally, the tree is printed in the Debug output window.

The first time in function *rt()* **CallTree** returns:

```
CallTree
Function rt( 4)
Sub t 1, 12, ToolBar(tb1) - Button, , Form(Win_1)
```

The program is executing function *rt()* with the parameter 4. The function was called from Sub t, which was called with the parameters 1, 12, an object - the Toolbar.Button object owned by Toolbar(tb1) -, empty (optional parameter declared As Variant), and the last parameter, a Form object with the name **Win_1**.

The next message box shows:

```
CallTree
Function rt( 3)
Function rt( 4)
Sub t 1, 12, ToolBar(tb1) - Button, , Form(Win_1)
```

Again, the program is currently executing the function *rt()*, now recursively called with parameter 3 from *rt()*, which itself was called earlier with parameter 4 from Sub t.

The third time:

```
CallTree
Function rt( 2)
Function rt( 3)
Function rt( 4)
Sub t 1, 12, ToolBar(tb1) - Button, , Form(Win_1)
```

The last time:

```
CallTree
Function rt( 1)
Function rt( 2)
Function rt( 3)
Function rt( 4)
Sub t 1, 12, ToolBar(tb1) - Button, , Form(Win_1)
```

Remarks

Especially for recursive subroutines **CallTree** occupies much stack memory, because of the nature of information; names and parameters as plain text. This could lead to a stack overflow. However, a stack overflow with **CallTree** will almost certainly create a stack overflow without **CallTree**, only some time later. Note that the performance decrease when using **CallTree** is significant.

In an EXE **CallTree** returns "".

A **Naked** subroutine is not included in the list. This is also true for code compiled with the **$StepOff** directive.

See Also

Naked, $StepOff

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

Tron, Troff Command

Purpose

Lists commands during program execution.

Syntax

Tron procedure

Troff

Description

The **Tron** command (TRACE On) causes each command that follows it to be redirected to the specified procedure. The procedure is executed before each command. **Troff** switches the redirection off. In the Tron procedure the following variables are available to inspect the program.

TraceLnr	Returns the current program line.
Trace$	Returns the source code text of the current line
TraceReg	Returns the procedssor register in the pseudo register variables **_EAX**, **_ECX**, etc. (8 registers)
SrcCode$(n)	Returns the specified source code line *n*.
ProcLnr(procname)	Returns the first line number of the specified subroutine (Procedure/Sub/Function).
ProcLineCnt(procname	Returns the number of lines of

)	the specified psubroutine.

Example

```
Local i%
OpenW # 1 : Debug.Show
Print "Test program"
Tron db
For i% = 1 To 5
  Print Sin(i%)
Next i%
Troff
Print "Program end"

Proc db
  Debug.Print Trace$
  // In the Debug output window each line (Trace$)
    is displayed.
EndProc
```

Remarks

The **Troff** turns the **Tron** off.

See Also

Debug, Trace, TraceLnr, TraceReg, SrcCode$, ProcLnr, ProcLineCnt

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

TraceLnr Function

Purpose

Returns the number of the command line to be executed next.

Syntax

% = **TraceLnr**

Description

TraceLnr returns an integer, inside the **Tron** *procedurename*, that contains the line number of the program line to be executed next. **Tron** *procedurename*, specifies a subroutine which will be invoked before execution of every command. The combination of **Tron** procedurename and **TraceLnr** is a very efficient way of looking for errors.

Example

See Trace$

Remarks

In a stand-alone program (EXE) the **Tron** command is ignored. **TraceLnr**, **ProcLnr**(p) and **ProcLineCnt**(p) are 0, **Trace$** and **SrcCode**(%) are "".

See Also

Tron, Debug, Trace, TraceLnr, TraceReg, SrcCode$, ProcLnr, ProcLineCnt, $StepOff

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

TraceReg

Purpose

Returns the address of memory block containing the processor register values.

Syntax

addr% = **TraceReg**

value = **TraceReg(***reg***)**

TraceReg(*reg***)** = value

addr, value:iexp

reg: a register, one of Eax, Ebx, Ecx, Edx, Ebp, Esp, Esi, Edi, Efl, Eip and the 16 bit register parts Ax, Bx, Cx, Dx, Bp, Sp, Si, Di, Fl, and the 8 bit register parts Al, Bl, Cl, Dl, Ah, Bh, Ch, Dh.

Description

TraceReg returns the address of a memory block containing the value of all processor register in the order edi esi esp ebp ebx edx ecx eax efl eip. To inspect the eax register you would use **LPeek**(**TraceReg** + 7*4), because eax is the seventh register in a row.

TraceReg(*reg*) only returns the value of one register in an appropriate pseudo variable. For instance Dim eax% = **TraceReg**(**Eax**).

TraceReg is used a **Tron** procedure, which is invoked before the next commandwill be executed next. **Tron** *procedurename*, specifies a subroutine which will be invoked before execution of every command.

Example

```
OpenW 1, 0, 0, 600, 500
Local j%
Global i1% = mAlloc(1000), i% = i1%
Tron p
. mov eax, 10
. mov [i%], eax
~1
Troff
~mFree(i1%)

Sub p
  Local d As New DisAsm
  d.ByteFlag = 1
  Local j%
  SetFont "courier new", 8
  Print Trace$
  d.Addr = TraceReg(Eip)
  For j = 1 To 5
    Exit If LPeek(d.Addr) %& 0xffffff == 0xb455ff
    Print d
  Next
  SetFont "Arial", 8, , 1
  Print "i ="; i; TraceLnr`Trace$
  EdShowLine TraceLnr - 1 : Delay .5
  If InStr(Trace$, "[i]") Then
    For j = 0 To 7
      Print {TraceReg + j * 4};
    Next
    Print
    Print "Eax ="; TraceReg(Eax)
    TraceReg(Eax) = 123
  EndIf
EndSub
```

The main program consists of two assembler instructions. The first one moves the value 10 to the register eax, the second moves the contents of eax to the variable i% (the ~1 makes sure, that the last used floating point register is cleared, not relevant here, though.)

The **Tron** procedure p prints the contents of the variable i% followed by the current line number and source code text of that line. The command **EdShowLine** shows the normal **Tron** arrow in front of the actual line. A small delay makes it possible to notice the current line.

Finally, if the source code line contains "[i%]", the value 123 is written as integer into memory, which address is obtained using **TraceReg**+7*4.
As a complete debugger, **Tron** needs access to the processor registers. **TraceReg** returns the address of the memory range, where for the actual processor registers are placed in. With **TraceReg**+7*4 the seventh register (0,1,2,3,4,5,6,eax) will be changed. As a result, 123 will placed in eax and thus in i%.

This example has been changed a little compared to the one presented in **EdShowLine**. In the **Tron** subroutine a **DisAsm** object is created and used to display the disassembly of the current line. After selecting a non-proportional font ("Courier New" 8 points) the next program line **Trace$** is displayed followed by a maximum of five lines of disassembly. The 'strange' **Exit If** compares the next assembler instruction to 'call dpt -76[ebp]'. This 3 byte instruction is generated between each program line when **$Step** is on. As a result, only the assembler code for the next to execute line is showed. The irrelevant code is ignored.

Remarks

In a stand-alone program (EXE) the **Tron** command is ignored. **TraceLnr**, **ProcLnr**(p) and **ProcLineCnt**(p) are 0, **Trace$** and **SrcCode**(%) are "".

See Also

Tron, Debug, Trace, TraceLnr, TraceReg, SrcCode$, ProcLnr, ProcLineCnt, $StepOff

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

SrcCode$ Function

Purpose

Returns the text of a source code line.

Syntax

$ = **SrcCode$**(line)

line:iexp

Description

Useable in a **Tron** procedure only.

Example

See Tron

Remarks

In Exe-Files **Tron** and **Troff** are ignored (no code will be generated), **TraceLnr**, **ProcLnr**(p) and **ProcLineCnt**(p) are 0, **Trace$** and **SrcCode**(%) are "".

See Also

Tron, EdShowLine, ProcLineCnt(), ProcLnr(), Trace TraceLnr, TraceReg

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

ProcLnr, ProcLineCnt Functions

Purpose

Return the starting line number and number of lines of a subroutine.

Syntax

% = **ProcLnr**(procname)

% = **ProcLineCnt**(procname)

Description

ProcLnr(procname)Returns the first line number of the specified subroutine (Procedure/Sub/Function).

ProcLineCnt(procname)Returns the number of lines of the specified subroutine.

Example

See Trace

Remarks

Used together with **Tron**.

See Also

Tron, EdShowLine, Trace, TraceLnr, TraceReg

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Mkn Functions

Purpose

Convert an integer to a string.

Syntax

string = **Mk1**[$](v, [,v1,..])

string = **Mk2**[$](v, [,v1,..])

string = **Mk3**[$](v, [,v1,..])

string = **Mk4**[$](v, [,v1,..])

string = **Mk5**[$](v, [,v1,..])

string = **Mk6**[$](v, [,v1,..])

string = **Mk7**[$](v, [,v1,..])

string = **Mk8**[$](v, [,v1,..])

Description

Mk1 converts one or more values in to string. **Mk2** converts one or more 2-byte (16-bit) values in a string, **Mk3** converts one or three-bytes of a value into a string, and so on.

Example

```
Print Mk1($41424344)     // D
Print Mk2($41424344)     // DC
Print Mk3($41424344)     // DCB
Print Mk4($41424344)     // DCBA
Print Mk5(Large $4142434445464748)     // HGFED
Print Mk6(Large $4142434445464748)     // HGFEDC
Print Mk7(Large $4142434445464748)     // HGFEDCB
Print Mk8(Large $4142434445464748)     // HGFEDCBA
```

Remarks

Mk1$() is the same as **Chr**$(), **Mk4**$() is the same as **Mkl**$(), and **Mk8**$() is the same as **MkLarge**$()

See Also

Cvn Functions, Mkl, Mki, Mkw, Mkd, Mks, MkCur, MkLarge

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

Cvn Functions

Purpose

Convert part of string to an integer.

Syntax

byte = **Cv1**(s$ [,offset% = 1])

short = **Cv2**(s$ [,offset% = 1])

long = **Cv3**(s$ [,offset% = 1])

long = **Cv4**(s$ [,offset% = 1])

large = **Cv5**(s$ [,offset% = 1])

large = **Cv6**(s$ [,offset% = 1])

large = **Cv7**(s$ [,offset% = 1])

large = **Cv8**(s$ [,offset% = 1])

Description

Cv1 converts one character form *s*$ into a byte. **Cv2** converts 2 characters, **Cv3** three characters, and so on. The *offset* parameter specifies the position within the string to use for converting. The default is 1, which is the start of the string.

The data type of the variable that holds the return value must be large enough to hold the value.

Example

```
Print Cv1("Hello GFA")     // Prints 72, 72 is the
  ASCII code of H
Print Cv2("Hello GFA")     // Prints 25928
Print Cv3("Hello GFA")     // Prints 7103816
Print Cv4("Hello GFA")     // Prints 18190443144
Print Cv5("Hello GFA")     // Prints 47856041300
Print Cv6("Hello GFA")     // Prints 35662932501832
Print Cv7("Hello GFA")     // Prints
  20020386278958408
Print Cv8("Hello GFA, 2") // Prints
  5064051968933913928
```

Remarks

Other functions convert (part of) a string to Currency (**CvCur**), Double (**Cvd**), Single (**Cvs**), Int32 (**Cvi**), and Word (**Cvw**).

The reverse of the Cv*n* functions are the Mk*n* functions (**Mk1**…**Mk8**)

See Also

Mkn Functions

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Cvd Function

Purpose

Converts the first eight characters in a string from binary to IEEE double format.

Syntax

double = Cvd(a$ [,offset%])

Description

Cvd() convert eight characters of a string into the IEEE Double format. **Cvd**() returns 0, if the length of the string is smaller than eight characters.

Example

```
OpenW 1
Local a$, a%, b$, c$, d$, e$
Open "Test.dat" for Random As # 1, Len = 21
Field # 1, 1 As a$, 4 As b$, 4 As c$, _
  4 As d$, 8 As e$
a$ = Chr$(123)
b$ = Mki$(1234)
c$ = Mkl$(12345678)
d$ = Mks$(1.23)
e$ = Mkd$(1.23)
Put # 1, 1
Get # 1, 1
Print Asc(a$)`Cvi(b$)`Cvl(c$)`Cvs(d$)`Cvd(e$)
// Prints: 123 1234 12345678 1.23 1.23
```

Remarks

Cvd() is the reverse function of **Mkd**().

See Also

Mkd$()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

CvdMbf Function

Purpose

Converts eight characters in a string from Microsoft Binary Format to IEEE double format.

Syntax

double **= CvdMbf**(a$ [,offset% = 1])

Description

As **CvsMbf** but only eight bytes and MBF-Double (In GWBASIC the four additional bytes are only filled with zero, i. e. the same 6 digits)

Remarks

CvdMbf() is the reverse function of **MkdMbf**().

Example

```
Print CvdMbf("GFABasic") //  Prints
 1.69857741858609e-09
```

See Also

CvsMbf(), MkdMbf()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Cvi Function

Purpose

Converts four characters in a string to a 32 bit integer.

Syntax

int32 **= Cvi**(a$ [,offset% = 1])

Description

Cvi takes four characters starting at *offset* in a string as a number. **Cvi**(a$) is equivalent to L**Peek**(V:a$). **Cvi** returns 0 if the string length is less than four.

Example

```
OpenW 1
Print Cvi("Hello GFA")
// prints 1819043144
Print Cvi(Mki$(24))
// Prints 24
Local a$ = Mki(100, 200, 300, 400)
Print Cvi(Mid$(a$, 1)), Cvi(Mid$(a$, 5)),
  Cvi(Mid$(a$, 9)) , Cvi(Mid$(a$, 13))
// Prints 100 200 300 400
Print Cvi(a$ , 5) // prints 200
```

Remarks

The order of the bytes depends on the processor. For 80x86/8 or 8088 processors LSB (least significant byte) is converted first and MSB (most significant byte) is converted last.

Cvi() is the reverse function of **Mki**$().

Cvi() is the same as **Cv4**() and **Cvl**()

See Also

Asc(), Cvl(), Cvs(), Cvd(), Chr$(), Mki$(), Mkl$(), Mks$(), Mkd$()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Cvl Function

Purpose

Converts four characters in a string to a 32 bit integer.

Syntax

long = **Cvl**(a$ [,offset% = 1])

Description

Cvl takes four characters starting at *offset* in a string as a number. **Cvl**(a$) is equivalent to LPeek(V:a$). **Cvl** returns 0 if the string length is less than four.

Example

```
OpenW 1
Print Cvl("Hello GFA")     // 1819043144
Print Cvl("Hello GFA", 2) // 1869376613
```

Remarks

Cvl() is the reverse function of **Mkl**$(). **Cvl**() is the same as **Cv4**() and **Cvi**().

See Also

Asc(), Cvi(), Cvs(), Cvd(), Chr$(), Mki$(), Mkl$(), Mks$(), Mkd$()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Cvs Function

Purpose

Converts four characters in a string from binary to IEEE single format.

Syntax

Single = **Cvs**(a$ [,offset% = 1])

Description

Cvs takes four characters starting at *offset* in a string as a number. **Cvs**(a$) is equivalent to **Single**{V:a$}. **Cvs** returns 0 if the string length is less than four.

Example

```
OpenW # 1
Print Cvs(Mks$(12.25)) // Prints 12.25
Local a$ = Chr$(123)
Local b$ = Mki$((1234))
Local c$ = Mkl$(12345678)
Local d$ = Mks$(1.23)
Local e$ = Mkd$(1.23)
Print Asc(a$)`Cvi(b$)`Cvl(c$)`Cvs(d$)`Cvd(e$)
// Prints 123 1234 12345678 1.23 1.23
```

Remarks

Cvs() is the reverse function of **Mks**$().

See Also

Asc(), Cvi(), Cvl(), Cvd(), Chr$(), Mki$(), Mkl$(), Mks$(), Mkd$()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

CvsMbf Function

Purpose

converts the four characters in a string from Microsoft Binary Float to IEEE single format.

Syntax

Single = **CvsMbf**(s$ [,offset% = 1])

Description

As an aid to read old GWBASIC Files containing binary floating point numbers - written with GWBASIC's Mks$() - in the Microsoft Binary Float (MBF) format there is now the Function CvsMbf() corresponding to Cvs()

Exmaple

```
Print CvsMbf("GFABasic") // Prints
  1.63709887045087e-19
Print Cvs("GFABasic")    // Prints 48.31863
```

Converts a number from a four byte string in MBF-Single format. (it has about 6 accurate digits, the rest are random).

See Also

Cvs, MksMbf$()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Cvw Function

Purpose

Converts two characters in a string to a 16 bit integer (word).

Syntax

word = **Cvw**(a$ [,offset% = 1])

Description

Cvw takes four characters starting at *offset* in a string as a number. **Cvw**(a$) is equivalent to DPeek(V:a$). **Cvw** returns 0 if the string length is less than two.

Example

```
OpenW 1
Print Cvw("Hello GFA")     // Prints 25928
Print Cvw("Hello GFA, 3") // Prints 25928
```

Remarks

Cvw() is the reverse function of **Mkw**$(). **Cvw**() is the same as **Cv2**().

See Also

Asc(), Cvi(), Cvs(), Cvd(), Chr$(), Mki$(), Mkl$(), Mks$(), Mkd$()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

CvCur Function

Purpose

Converts eight characters in a string to a Currency value.

Syntax

Currency **= CvCur**(s$ [,offset%])

Description

The *offset* parameter specifies the position within the string to use for converting. The default is 1, which is the start of the string.

Example

```
Print CvCur("Hello GFA") // prints
 506405196893391.3928
```

Remarks

Other functions convert (part of) a string to integer (**Cvn**), Double (**Cvd**), Single (**Cvs**), Int32 (**Cvi**), and Word (**Cvw**).

The reverse of the **CvCur** is **MkCur**

See Also

MkCur

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

CvLarge Function

Purpose

Converts eight characters in a string to a 64 bit integer.

Syntax

large = **CvLarge**(a$ [,offset% = 1])

Description

CvLarge takes eight characters starting at *offset* in a string as a number. **CvLarge** is equivalent to **Cv8**. **CvLarge** returns 0 if the string length is less than eight.

Example

```
OpenW 1
Print CvLarge("Hello GFA")     //
  5064051968933913928
Print CvLarge("Hello GFA", 2) //
  4703525065468963941
```

Remarks

CvLarge() is the reverse function of **MkLarge**$().

See Also

Asc(), Cvi(), Cvs(), Cvd(), Chr$(), Mki$(), Mkl$(), Mks$(), Mkd$()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

MkCur Function

Purpose

converts a Currency (64-bit) expression to eight characters.

Syntax

$ = **MkCur**[$](x [,x1,..])

x, x1,..: Currency

Description

Creates an eight characters long string from a number internally stored in IEEE double format.

Example

```
OpenW # 1
Print MkCur$(2.1, 3.4)
```

Remarks

See Also

Cvn Functions, Mkl, Mki, Mkw, Mkd, Mks, MkCur, MkLarge

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

Mkd$ Function

Purpose

converts a 64-bit floating point expression to eight characters.

Syntax

$ = **Mkd**[$](x [,x1,..])

x, x1, ...: Double

Description

Creates an eight characters long string from a number internally stored in IEEE double format.

Example

```
OpenW 1
Print MkdMbf$(2.1)
Print MkdMbf$(2.1, 6.4)
Print Mkd(2.1)
Print Mkd(2.1, 6.4)
```

Remarks

See Also

Cvn Functions, Mkl, Mki, Mkw, Mkd, Mks, MkCur, MkLarge

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

MkdMbf$ Function

Purpose

This function is used to convert floating point numbers in Microsoft Binary Float (MBF) into an 8-byte string.

Syntax

$ = **MkdMbf**$(x [, x1,..])

x,x1,..floating point value in Microsoft Binary Float format

Description

As an aid to real old GWBASIC files containing binary floating point numbers written with GWBASIC this function is provided to convert an MBF-floating point number into an 8-byte string.

Remarks

This is the reverse of the function **CvdMbf**()

Example

```
OpenW 1
Print MkdMbf$(2.1)
Print MkdMbf$(2.1, 6.4)
Print Mkd(2.1)
Print Mkd(2.1, 6.4)
```

See Also

CvsMbf, CvdMbf, MksMbf$

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

Mki Function

Purpose

converts a 32-bit integer expression to a four character string.

Syntax

$ = **Mki**[$](x [,x1,..])

x, x1,..: Integer

Description

Creates a four character long string from an integer. Additional arguments increases the size of the string with a multiple of four.

Example

```
Local a$, b$, c$, d$, e$
OpenW # 1
Open "C:\Test.DAT" for Random As # 1, Len = 19
Field # 1, 1 As a$, 2 As b$, 4 As c$, 4 As d$, 8
  As e$
a$ = Chr$(123)
b$ = Mkw$(1234)
c$ = Mki$(12345678)
d$ = Mks$(1.23)
e$ = Mkd$(1.23)
Put # 1, 1
//
Get # 1, 1
```

```
Print Asc(a$)`Cvi(b$)`Cvl(c$)`Cvs(d$)`Cvd(e$)
Close # 1
Kill "c:\test.dat"
```

prints 123 1234 12345678 1.23000019.. 1.23

Remarks

Mki$() is the reverse function of **Cvi**().

See Also

Cvn Functions, Mkl, Mki, Mkw, Mkd, Mks, MkCur, MkLarge

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

Mkl Function

Purpose

converts a 32-bit integer expression to a four character string.

Syntax

$ = **Mkl**[$](x [,x1,..])

x, x1,..: Integer

Description

Creates a four character long string from an integer. Additional arguments increases the size of the string with a multiple of four.

Example

```
Local a$, b$, c$, d$, e$
OpenW # 1
Open "C:\Test.DAT" for Random As # 1, Len = 19
Field # 1, 1 As a$, 2 As b$, 4 As c$, 4 As d$, 8
  As e$
a$ = Chr$(123)
b$ = Mkw$(1234)
c$ = Mki$(12345678)
d$ = Mks$(1.23)
e$ = Mkd$(1.23)
Put # 1, 1
//
Get # 1, 1
```

```
Print Asc(a$)`Cvi(b$)`Cvl(c$)`Cvs(d$)`Cvd(e$)
Close # 1
Kill "c:\test.dat"
```

prints 123 1234 12345678 1.23000019.. 1.23

Remarks

Mkl$() is the reverse function of **Cvl**().

See Also

Cvn Functions, Mkl, Mki, Mkw, Mkd, Mks, MkCur, MkLarge

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

MkLarge Function

Purpose

converts a 64-bit integer expression to a eight character string.

Syntax

$ **= MkLarge**[$](x [,x1,..])

x, x1,..: Integer

Description

Creates a eight character long string from a large integer. Additional arguments increases the size of the string with a multiple of eight.

Example

```
Dim s As Large = CvLarge("abcdefgh")
Print MkLarge(s, s)
```

Prints abcdefghabcdefgh

Remarks

MkLarge$() is the reverse function of **CvLarge**().

See Also

Cvn Functions, Mkl, Mki, Mkw, Mkd, Mks, MkCur, MkLarge

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

Mks Function

Purpose

converts a 32-bit floating point (Single) expression into a four characters string.

Syntax

$ = **Mks**[$](x [,x1,..])

x, x1, ..: Single

Description

Creates a four character long string from a number internally stored in IEEE single format.

Example

```
Dim s1 As Single = Cvs("abcd")
Dim s2 As Single = CvsMbf("abcd")
Print s1, Hex(LPeek(V:s1), 4)
Print s2, Hex(LPeek(V:s2), 4)
Print Mks(s1, s1)
Print MksMbf$(s2, s2)
```

Remarks

Mks() is the reverse function of **Cvs**().

See Also

Cvn Functions, Mkl, Mki, Mkw, Mkd, Mks, MkCur, MkLarge

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

MksMbf$ Function

Purpose

This function is used to convert floating point numbers in Microsoft Binary Float (MBF) into an 4-byte string.

Syntax

$ = **MksMbf**$(x [, x1,..])

x,x1,...	*:floating point value in Microsoft Binary Float format*

Description

As an aid to real old GWBASIC files containing binary floating point numbers written with GWBASIC this function is provided to convert an MBF-floating point number into an 8-byte string.

Remarks

This is the reverse of the function **CvsMbf**()

Example

```
Dim s1 As Single = Cvs("abcd")
Dim s2 As Single = CvsMbf("abcd")
Print s1, Hex(LPeek(V:s1), 4)
Print s2, Hex(LPeek(V:s2), 4)
Print Mks(s1, s1)
Print MksMbf$(s2, s2)
```

See Also

CvsMbf, CvdMbf, MksMbf$

{Created by Sjouke Hamstra; Last updated: 28/02/2017 by James Gaite}

Mkw Function

Purpose

converts a 16-bit integer expression into a two characters string.

Syntax

$ = **Mkw**[$](x [,x1,..])

x, x1, ..: Single

Description

Creates a two character long string from a number.

Example

```
Dim s As Short = Cvw("ab")
Print Mkw(s, s)
```

Prints: abab

Remarks

Mkw() is the reverse function of **Cvw**().

See Also

Cvn Functions, Mkl, Mki, Mkw, Mkd, Mks, MkCur, MkLarge

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

Format Function

Purpose

Returns a **String** containing an expression formatted according to instructions contained in a format expression.

Syntax

$ = **Format**(*expression*[, *format*])

Description

Format() is a general conversion function for which you have almost total domination of its behavior. **Format** can format numerical, string, and date expressions. Other expressions are first converted to double or string using the regional settings dependent functions **CDbl**() and **CStr**().

Sections - A user-defined format expression can have several sections separated by semicolons (**;**). A format expression for strings can have one section or two sections separated by a semicolon. If you use one section inly, the format applies to all string data. If you use two sections, the first section applies to string data, the second to **Null** values and zero-length strings ("").

General

The following characters apply to all user-defined formats.

- + $ ( )	Display a literal character. To display a character other than one of those listed, precede it with a

	backslash (\) or enclose it in double quotation marks (" ").
(\)	Display the next character in the format string. To display a character that has special meaning as a literal character, precede it with a backslash (\). The backslash itself isn't displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (\\\\).Examples of characters that can't be displayed as literal characters are the date-formatting and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y, / and :), the numeric-formatting characters (#, 0, %, E, e, comma, and period), and the string-formatting characters (@, &, <, >, and !).
("ABC")	Display the string inside the double quotation marks (" "). To include a string in ***format*** from within code, you must use **Chr(**34**)** to enclose the text (34 is the character code for a quotation mark ("")).

Numbers

A user-defined format expression for numbers can have from one to four sections separated by semicolons. If the *format* argument contains one of the named numeric formats, only one section is allowed. With one section only, the *format* expression applies to all values. With multiple sections, the first section applies to positive values and zeros, the second to negative values, and the third to zeros. With four sections the fourth is reserved for **Null** values (Variant).

The following example has two sections: the first defines the format for positive values and zeros; the second section defines the format for negative values "$#,##0;($#,##0)".

If you include semicolons with nothing between them, the missing section is printed using the format of the positive value. For example, the following format displays positive and negative values using the format in the first section and displays "Zero" if the value is zero "$#,##0;;\Z\e\r\o".

Create **user-defined** numeric formats using any of the following characters.

0	Digit placeholder. Display a digit or a zero. If the expression has a digit in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position. If the number has fewer digits than there are zeros (on either side of the decimal) in the format expression, display leading, or trailing zeros. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, round the number to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, display the extra digits without modification.
#	Digit placeholder. Display a digit or nothing. This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has the same or fewer digits than there are # characters on either side of the decimal separator. `Print Format$(3.14, "###.###;;")    // "  3,14"`

```
Print Format$(3.14, "###.##0;;")     // "
  3,140"
Print Format$(3.14, "###.###**;;")   //
  "**3,14*"
Print Format(0.14, "###.###;;")      //
  "    ,14"
Print Format(0.14, "##0.##0;;")      // "
  0,140"
Print Format(0.14, "###.###**;;")    //
  "***,14*"
```

. Decimal placeholder. In some locales, a comma is used as the decimal separator. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only number signs to the left of this symbol, numbers smaller than 1 begin with a decimal separator. To display a leading zero displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator. The actual character used as a decimal placeholder in the formatted output depends on the **Mode Format** setting, regional system setting, or **Mode Lang**.

, Thousand separator. In some locales, a period is used as a thousand separator. The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Standard use of the thousand separator is specified if the format contains a thousand separator surrounded by digit placeholders (**0** or **#**). Two adjacent thousand separators or a thousand separator immediately to the left of the decimal separator (whether or not a decimal is specified) means "scale the number by dividing it by 1000, rounding as needed." For example, you can use

	the format string "##0,," to represent 100 million as 100. Numbers smaller than 1 million are displayed as 0. Two adjacent thousand separators in any position other than immediately to the left of the decimal separator are treated simply as specifying the use of a thousand separator. The actual character used as the thousand separator in the formatted output depends on the Number Format recognized by your system.
%	Percentage placeholder. The expression is multiplied by 100. The percent character (**%**) is inserted in the position where it appears in the format string.
E- E+ e- e+	Scientific format. If the format expression contains at least one digit placeholder (**0** or **#**) to the right of E-, E+, e-, or e+, the number is displayed in scientific format and E or e is inserted between the number and its exponent. The number of digit placeholders to the right determines the number of digits in the exponent. Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a minus sign next to negative exponents and a plus sign next to positive exponents.

Predefined named number formats.

General Number	Display number with no thousand separator.
Currency	Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator. Output is based on system locale settings.

Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Standard	Display number with thousand separator, at least one digit to the left and two digits to the right of the decimal separator.
Percent	Display number multiplied by 100 with a percent sign (**%**) appended to the right; always display two digits to the right of the decimal separator.
Scientific	Use standard scientific notation.
Yes/No	Display No if number is 0; otherwise, display Yes.
True/False	Display **False** if number is 0; otherwise, display **True**.
On/Off	Display Off if number is 0; otherwise, display On.

Date and Time

User-defined date and time formats. Use any of the following characters.

:	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by **Mode Format** or your system settings.
/	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are

	formatted. The actual character used as the date separator in formatted output is determined by **Mode Format** or your system settings.
c	Display the date as ddddd and display the time as ttttt, in that order. Display only date information if there is no fractional part to the date serial number; display only time information if there is no integer portion.
d	Display the day as a number without a leading zero (1 - 31).
dd	Display the day as a number with a leading zero (01 - 31).
ddd	Display the day as an abbreviation (Sun - Sat).
dddd	Display the day as a full name (Sunday - Saturday).
ddddd	Display the date as a complete date (including day, month, and year), formatted according to your system's short date format setting. For Microsoft Windows, the default short date format is m/d/yy.
dddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the long date setting recognized by your system. For Microsoft Windows, the default long date format is mmmm dd, yyyy.
w	Display the day of the week as a number (1 for Sunday through 7 for Saturday).
ww	Display the week of the year as a number (1 - 53).
m	Display the month as a number without a leading zero (1 - 12). If m immediately follows

	h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01 - 12). If m immediately follows h or hh, the minute rather than the month is displayed.
M	Display the month as a number without a leading zero (1 - 12).
MM	Display the month as a number with a leading zero (01 - 12).
mmm	Display the month as an abbreviation (Jan - Dec) (also **MMM**).
mmmm	Display the month as a full month name (January - December) (also **MMMM**).
q	Display the quarter of the year as a number (1 - 4).
y	Display the day of the year as a number (1 - 366).
yy	Display the year as a 2-digit number (00 - 99).
yyyy	Display the year as a 4-digit number (100 - 9999).
h	Display the hour as a number without leading zeros (0 - 23) (also **H**).
hh	Display the hour as a number with leading zeros (00 - 23) (also **HH**).
n	Display the minute as a number without leading zeros (0 - 59).
nn	Display the minute as a number with leading zeros (00 - 59).
s	Display the second as a number without leading zeros (0 - 59).
ss	Display the second as a number with leading zeros (00 - 59).

ttttt	Display a time as a complete time (including hour, minute, and second), formatted using the time separator defined by the time format recognized by your system. A leading zero is displayed if the leading zero option is selected and the time is before 10:00 A.M. or P.M. For Microsoft Windows, the default time format is h:mm:ss.
AM/PM	Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour between noon and 11:59 P.M.
am/pm	Use the 12-hour clock and display a lowercase AM with any hour before noon; display a lowercase PM with any hour between noon and 11:59 P.M.
A/P	Use the 12-hour clock and display an uppercase A with any hour before noon; display an uppercase P with any hour between noon and 11:59 P.M.
a/p	Use the 12-hour clock and display a lowercase A with any hour before noon; display a lowercase P with any hour between noon and 11:59 P.M.
AMPM	Use the 12-hour clock and display the AM string literal as defined by your system with any hour before noon; display the PM string literal as defined by your system with any hour between noon and 11:59 P.M. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as defined by your system settings. For Microsoft Windows, the default format is AM/PM.

Predefined named date/time formats.

General Date	Display a date and/or time. For real numbers, display a date and time, for example, 4/3/93 05:34 PM. If there is no fractional part, display only a date, for example, 4/3/93. If there is no integer part, display time only, for example, 05:34 PM. Date display is determined by your system settings (not **Mode Format**).
Long Date	Display a date according to your system's long date format.
Medium Date	Display a date using the medium date format appropriate for the language version of the host application.
Short Date	Display a date using your system's short date format.
Long Time	Display a time using your system's long time format; includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minutes and the AM/PM designator.
Short Time	Display a time using the 24-hour format, for example, 17:45.

Note - Format(date) without a format string returns the "General Date" or "c".

Strings

User-defined string formats. Use any of the following characters.

@	Character placeholder. Display a character or a space. If the string has a character in the position where the at symbol (@) appears in the

	format string, display it; otherwise, display a space in that position. Placeholders are filled from right to left unless there is an exclamation point character (!) in the format string.
&	Character placeholder. Display a character or nothing. If the string has a character in the position where the ampersand (&) appears, display it; otherwise, display nothing. Placeholders are filled from right to left unless there is an exclamation point character (!) in the format string.
<	Force lowercase. Display all characters in lowercase format.
>	Force uppercase. Display all characters in uppercase format.
!	Force left to right fill of placeholders. The default is to fill placeholders from right to left.

```
Print Format("GFA BASIC32", "&&&")  // GFA
Print Format("GFA BASIC32", "&&&<") // gfa basic32
Print Format("Test", "**&&&&&&&&")  // Test****
Print Format("Test", "**&&&&&&&&!") // ****Test
```

Example

```
Debug.Show
Local Date MyTime, MyDate
MyTime = #17:04:23#
MyDate = #04/10/2008#
' Returns current system time in the system-
  defined long time format.
Trace Format(Time, "Long Time")
' Returns current system date in the system-
  defined long date format.
Trace Format(Date, "Long Date")
Trace Format(MyTime, "h:m:s")            //
  "17:4:23".
Trace Format(MyTime, "hh:mm:ss AM/PM")   //
  "05:04:23 PM".
Trace Format(MyDate, "dddd, mmm d yyyy")//
  "Thursday, Apr 10 2008".
' If format is not supplied, a string is returned.
Trace Format(23)    //"23"
' User-defined formats.
Trace Format(5459.4, "##,##0.00")    // "5.459,40".
Trace Format(334.9, "###0.00")       // "334.90".
Trace Format(5, "0.00%")             // "500.00%".
```

Remarks

If you try to format a number without specifying *format*, **Format** provides functionality similar to the **Str** function, although it is internationally aware. However, positive numbers formatted as strings using **Format** don’t include a leading space reserved for the sign of the value; those converted using **Str** retain the leading space.

The *format* string can not exceed 1023 characters.

See Also

Str, CStr, CDbl, Mode

{Created by Sjouke Hamstra; Last updated: 06/10/2014 by James Gaite}

DateSerial Function

Purpose

Returns a **Variant** (**Date**) for a specified year, month, and day.

Syntax

v = **DateSerial**(*year*, *month*, *day*)

NOTE: The **DateSerial** function can be entered as above but will automatically be converted into **DateSerial**((*year*,*month*,*day*,)). The parameter after the final comma appears to serve no purpose and a Syntax Error is returned if a value is entered.

v: Variant
year, month, day: iexp

Description

To specify a date, such as December 31, 1991, the range of numbers for each **DateSerial** argument should be in the accepted range for the unit; that is, 1-31 for days and 1-12 for months. However, you can also specify relative dates for each argument using any numeric expression that represents some number of days, months, or years before or after a certain date.

The following example uses numeric expressions instead of absolute date numbers. Here the **DateSerial** function returns a date that is the day before the first day (1 - 1), two months before August (8 - 2), 10 years before 1990 (1990 - 10); in other words, May 31, 1980.

```
DateSerial((1990 - 10, 8 - 2, 1 - 1, ))
```

For the *year* argument, values range from 100 to 9999, inclusive.

When any argument exceeds the accepted range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 35 days, it is evaluated as one month and some number of days, depending on where in the year it is applied.

Example

```
Print DateSerial((1999 - 40, 10 - 12., 30 - 44, ))
  // prints 16/09/58
```

Remarks

The output of the **DateSerial** function is not affected by **Mode Date** or **Mode Time** and separates the date elements with the '/' symbol. To get around this problem and standardise your date format, put the **DateSerial** function inside **Date$**() or **DateTime$**() as below:

```
Mode Date "."
Print DateSerial((1900, 10, 1, ))
Print Date$(DateSerial((1900, 10, 1,)))
```

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(),

DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

DateValue Function

Purpose

Returns the passed time since 1 January 1899.

Syntax

var = **DateValue**(exp)

var	*: variable*
exp	*: aexp*

Description

This function converts *exp* and returns a date value in *var*. The conversion uses the VarDateFromString API and so takes into account the Regional settings of the system rather than the current GB **Mode** setting (for a GB **Mode** compliant function, see ValDate).

The expression *exp* can be a string, date, or date literal. The date literal must use a period (or full stop) separator for date (25.12.2018) regardless of Mode settings.

The value returned to *var* depends on the variable type for the return value but is **Date** by default: when *var* is a **Single** or **Double** the number of days since 1 January 1899 is returned; where *var* is of type **Integer**, the date part without the time is returned (see Known Issues below); and if *var* is a **String**, the date and time are returned as a string.

Example

```
Local da As Date, db As Double, i As Int, s As
  String
da = DateValue("25 Jan 2019 11:42") : Print da  //
  25/01/2019 11:42:00
db = DateValue("25 Jan 2019 11:42") : Print db  //
  43490.4875
i = DateValue("25 Jan 2019 11:42") : Print i    //
  43490 - See Known Issues re Integers
s = DateValue("25 Jan 2019 11:42") : Print s    //
  25/01/2019 11:42:00
Print DateValue(#25.01.2019 11:42:00#)          //
  25/01/2019 11:42:00
Print VarType(DateValue(#25.01.2019 11:42:00#)) //
  7 = Date
```

Remarks

The base year, at least for Windows 98, is 1930. This means that years specified with only two digits are interpreted a based to 1930. A year of "29" means 2029, and "31" means 1931. Since this is OLE dependent, located in oleaut32.dll, it cannot be adjusted.

The output of the **DateValue** function is not affected by **Mode Date** or **Mode Time** and separates the date elements with the '/' symbol. To get around this problem and standardise your date format, put the **DateValue** function inside **Date$**() or **DateTime$**() as below:

```
Mode Date "."
Print DateValue(Date)
Print Date$(DateValue(Date))
```

Known Issues

When using **DateValue** to return a converted value to an **Integer**, if the Time element is greater than 12:00 noon then the date value is rounded up rather than truncated, resulting in the wrong date being returned. To get around this, you can use the **Trunc** function as shown in the example below:

```
Local i As Integer
i = DateValue("25 Jan 2019 11.42") : Print i
      // 43490
i = DateValue("25 Jan 2019 12.42") : Print i
      // 43491
i = Trunc(DateValue("25 Jan 2019 12.42")) : Print
 i    // 43490
```

[Reported by Sjouke Hamstra, 30/01/2019]

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 02/02/2019 by James Gaite}

Hex$ Function

Purpose

Converts an integer expression to hexadecimal representation.

Syntax

string = **Hex**[**$**](m[,n])

Description

After conversion the hexadecimal representation of integer expression m is returned as a plain string.

The parameter n is optional and determines how many places should be used to represent the number. If n is greater than the number of places needed to represent m the converted number is padded with leading zeros.

Example

```
Debug.Show
Trace Hex$(25)       // Prints 19
Trace Hex$(1001, 6) // Prints 0003E9
```

See Also

Bin$(), Oct$(), Dec$()

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

Oct Function

Purpose

Converts an integer expression to octal representation.

Syntax

$ = **Oct**[$](m[,n])

m, n:integer expression

Description

After conversion the octal representation of integer expression m is returned as a plain string. The parameter n is optional and determines how many places should be used. If n is greater than the number of places needed to represent m the converted number is padded with leading zeros.

Example

```
Debug.Show
Trace Oct$(17)      //prints 21
Trace Oct$(25, 6)   //prints 000031
```

Remarks

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

See Also

String, Bin$(), Hex$(), Dec$()

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Bin Function

Purpose

Converts an integer expression to a binary string representation.

Syntax

Bin[$](m[,n])

Description

After conversion the binary representation of integer expression m is returned as a plain string.

The parameter n is optional and determines how many places (1 to 33) should be used to represent the number. If n is greater than the number of places needed to represent m the converted number is padded with leading zeros.

Example

```
OpenW # 1
Print Bin$(17)     // Prints 10001
Print Bin$(25, 6)  // Prints 011001
```

Remarks

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

See Also

String, Oct$(), Hex$(), Dec$()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Dec$ Function

Purpose

converts an integer expression to decimal representation.

Syntax

Dec[$](m[,n]

Description

Dec[$](m[,n] converts the integer expression m into decimal representation. This is a base 10 number system with digits from 0 to 9. The optional parameter n specifies how many places should be used. If n is greater than the number of places needed by m, the number is padded with leading zeros.

Example

```
OpenW # 1
Print Dec$(25)     // Prints 25
Print Dec(123, 6) // Prints 000123
```

Remarks

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

See Also

String, Bin$(), Hex$(), Oct$()

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

Base Function

Purpose

Returns a string representing a number using a specific base.

Syntax

string = **Base**[$](value [, :] radix)

value: iexp
radix: character (0 - 9, A - Z)

string = **Base**$(**&** radix : value, newradix)

value: word
radix, newradix: character (0 - 9, A - Z)

Description

Base$() converts the digits of *value* to a character string and stores the result (up to 33 bytes) in *string*. The *radix* argument specifies the base of *value*, which must be a character in the range 0 - 9 and A - Z. For example:

```
Print Base$(21286:Z) // prints GFA
Print Base$(21286:9) // prints 21286
```

When used in this way, where a numeric value is separated with a colon and followed with a radix, the radix is limited to 2-9 and A-Z.

When used with comma, radix may be chosen from 2-36. For example

```
Print Base$(21286, 2)  // prints 101001100100110
Print Base$(21286, 8)  // prints 51446
Print Base$(21286, 10) // prints 21286
Print Base$(21286, 16) // prints 5326
```

These are the general forms for **Bin**$(), **Oct**$() **Dec**$(), and **Hex**$().

To convert a word into a number the following format is used:

Base$(**&** radix **:** word, newradix)

radix specifies the base of the word that is to be converted to *newradix*. *newradix* can be any character between 0-9 and A-Z, but it can also be a number in the range from 0-36. For instance, the radix Z equals ',36'.

Example

```
OpenW # 1
Print Base$(21286:Z)    // prints GFA
// The inverse...
Print Base$(&Z:GFA, 10) // prints 21286
```

See Also

Bin$, Oct$, Dec$, Hex$

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Str Function

Purpose

Converts a numeric expression into a string.

Syntax

$ = **Str**[$](x [,m, n])

x:aexp

m, n:integer expression

Description

Str$(x, m) converts x into a string of m length. If m is greater than the number of characters needed to represent x, the string is padded with leading spaces. If m is smaller than the number of characters needed to represent x, the string is truncated from the right.

Str$(x, m, n) converts x into a string of m length with n decimal places. The last decimal place is rounded off. Out of the total length m, n+1 places are reserved (n places for the decimal part and one place for the decimal point).

With positive expressions **Str** adds a space in front of the number. With negative values a minus is added. The additional space in front of positive values is a VB quirk and is mimicked by GFA-BASIC 32. To prevent the space for positive numbers use **Mode StrSpace 0**.

Example

```
Debug.Show
Trace Str$(3 * 4 + 2)       //  Prints " 14"
Local a$ = Str$(3 * 4 + 2)
Mode StrSpace 0
Trace a$                    //  Prints "14"
Trace Str$(123.456, 7)      //  Prints 123.456
Trace Str$(123.456, 9)      //  Prints   123.456
Trace Str$(123.456, 5)      //  Prints 123.4
Trace Str$(123.456, 7, 3) //  Prints 123.456
Trace Str$(123.456, 7, 5) //  Prints 3.45600
Trace Str$(123.456, 7, 2) //  Prints 123.46
Trace Str$(123.456, 9, 3) //  Prints 123.456
```

Remarks

The **Print** [**#**] commands use the **Str**() function internally to convert a numeric expression to a printable string. Therefore, **Print** adds a space in front of a positive value as well. The **Mode StrSpace 0** prevents the adding of a space.

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

See Also

String, Dec$(), Hex$(), Oct$(), CStr, Using, Mode, Format, sprintf

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

sprintf Function

Purpose

Returns a string with formatted data.

Syntax

$ = sprintf[$]**(***format$* [**,** *argument*] ... **)**

format:sexp
argument, …:aexp

Description

The **sprintf** function formats and stores a series of characters and values in string. Each *argument* (if any) is converted and output according to the corresponding format specification in *format*.

Character combinations consisting of a backslash (****) followed by a letter or by a combination of digits are called “escape sequences.” To represent a newline character, single quotation mark, or certain other characters in a character constant, you must use escape sequences. An escape sequence is regarded as a single character and is therefore valid as a character constant. Escape sequences are typically used to specify actions such as carriage returns and tab movements on terminals and printers. They are also used to provide literal representations of nonprinting characters and characters that usually have special meanings, such as the double quotation mark (**"**).

Escape sequence for **sprintf** are:

"\a"	Chr(7) Bell alert
"\b"	Chr(8) Backspace
"\e"	Chr(27) Escape
"\t"	Chr(9) Vertical tab
"\n"	Chr(10) New line
"\r"	Chr(13) Carriage return
"\f"	Chr(12) Formfeed
"\v"	Chr(11) Vertical tab.
"\\"	Backslash
"\%"	% an expansion
"\#nnn"	ASCII character in decimal notation (\#27 is similar to Chr(27)).
"\ooo"	ASCII character in octal notation (\033 is similar to **Chr**(0o033); each o represents only one octal digit (0..7)).
"\xhhh"	ASCII character in hexadecimal notation (\x1b is similar to Chr(0x1b), each h represents one hexadecimal digit (0..9a..fA..F)).
"\ "	\Space: this is no sequence like the others before, its purpose is to end \character codes: "\10\ 33" "\b33", but "\1033" results in "C3".

Format specifications

Format specifications always begin with a percent sign (**%**) and are read left to right. When **sprintf** encounters the first format specification (if any), it converts the value of the first argument after *format* and outputs it accordingly. The second format specification causes the second argument to be converted and output, and so on. If there are more arguments than there are format specifications, the extra arguments are ignored. The results are undefined if there are not enough arguments for all the format specifications.

A format specification, which consists of optional and required fields, has the following form:

%[flags] [width] [**.** precision] [{**h** | **l** | **L**}]type

h | **l** | **L** Optional prefixes to *type*-that specify the size of *argument*

h - short

I - long int

L - Large, int64

```
Print sprintf("%Li", Large 120986754678)  //
 120986754678
Print sprintf("%Ii", Large 120986754678)  //
 727670390
```

Type	**Meaning**
%d	Signed decimal integer (Int)
%i	Signed decimal integer (Int)
%x	Unsigned hexadecimal integer, using "abcdef". (Int)
%X	Unsigned hexadecimal integer, using "ABCDEF". (Int)
%o	Unsigned octal integer (Int)
%f	Signed value having the form [-]*dddd***.***dddd*, where *dddd* is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision. (Double)
%e	Signed value having the form [-]*d***.***dddd* **e** [*sign*]*ddd* where *d* is a single decimal digit, *dddd* is

	one or more decimal digits, *ddd* is exactly three decimal digits, and *sign* is + or -. (Double)
%E	Identical to the **e** format except that **E** rather than **e** introduces the exponent. (Double)
%g	Signed value printed in **f** or **e** format, whichever is more compact for the given value and precision. The **e** format is used only when the exponent of the value is less than -4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it. (Double)
%G	Identical to the **g** format, except that **E**, rather than **e**, introduces the exponent (where appropriate). (Double)
%s	String. Characters are printed up to the first null character or until the *precision* value is reached.
%c	Character

Flags

The first optional field of the format specification is *flags*. A flag directive is a character that justifies output and prints signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag directive may appear in a format specification.

Specification	Meaning
-	Left align the result within the given field width. Right align.
+	Prefix the output value with a sign (+ or -) if the output value is of a signed type. Sign appears only for negative signed values (-).
0	If width is prefixed with 0, zeros are

	added until the minimum width is reached. If 0 and - appear, the 0 is ignored. If 0 is specified with an integer format (i, u, x, X, o, d) the 0 is ignored.	No padding.
blank (' ')	Prefix the output value with a blank if the output value is signed and positive; the blank is ignored if both the blank and + flags appear.	No blank appears.
#	When used with the o, x, or X format, the # flag prefixes any nonzero output value with 0, 0x, or 0X, respectively.	No blank appears.
	When used with the e, E, or f format, the # flag forces the output value to contain a decimal point in all cases.	Decimal point appears only if digits follow it.
	When used with the g or G format, the # flag forces the output value to contain a decimal point in all cases and prevents the truncation of trailing zeros. Ignored when used with c, d, i, u, or s.	Decimal point appears only if digits follow it. Trailing zeros are truncated.

```
Print sprintf("%+i", -255) // Prints -255
```

Width

The second optional field of the format specification is the *width* specification. The *width* argument is a nonnegative decimal integer controlling the minimum number of characters printed. If the number of characters in the output value is less than the specified width, blanks are added to the left or the right of the values - depending on whether the - flag (for left alignment) is specified - until the minimum width is reached. If *width* is prefixed with 0, zeros are added until the minimum width is reached (not useful for left-aligned numbers).

The width specification never causes a value to be truncated. If the number of characters in the output value is greater than the specified width, or if *width* is not given, all characters of the value are printed (subject to the precision specification).

```
Debug.Show
Debug.Print sprintf("%6i", 0)   // "     0"
Debug.Print sprintf("%6s", "xx")// "    xx"
```

Precision

The third optional field of the format specification is the *precision* specification. It specifies a nonnegative decimal integer, preceded by a period (**.**), which specifies the number of characters to be printed, the number of decimal places, or the number of significant digits. Unlike the width specification, the precision specification can cause either truncation of the output value or rounding of a floating-point value.

Type	Meaning
c, C	The precision has no effect. Character is printed.
d, i, o, x, X	The precision specifies the minimum number of digits to be printed. If the number of digits in the argument is less than precision, the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds precision. Default precision is 1.
e, E	The precision specifies the number of digits to be printed after the decimal point. The last printed digit is rounded. Default precision is 6; if precision

	is 0 or the period (.) appears without a number following it, no decimal point is printed.
f	The precision value specifies the number of digits after the decimal point. If a decimal point appears, at least one digit appears before it. The value is rounded to the appropriate number of digits. Default precision is 6; if precision is 0, or if the period (.) appears without a number following it, no decimal point is printed.
g, G	The precision specifies the maximum number of significant digits printed. Six significant digits are printed, with any trailing zeros truncated.
s, S	The precision specifies the maximum number of characters to be printed. Characters in excess of precision are not printed. Characters are printed until a null character is encountered.

```
Print sprintf("%+.8i", -255) // "-00000255"
Print sprintf("%+.4e", -255) // "2.5500e+002"
```

Example

```
Print sprintf("%d is in octal %o", 255, 255)
Print sprintf("%d is in int %i", 255, 255)
Print sprintf("%d is in hexadecimal %x", 255, 255)
Print sprintf("%d is in hexadecimal %X", 255, 255)
Print sprintf("%d is in double %f", 255, 255)
Print sprintf("%d is in double %e", 255, 255)
Print sprintf("%d is in double %E", 255, 255)
Print sprintf("%d is in compact double %g", 255,
 255)
```

Prints:

```
255 is in octal 377
255 is in int 255
255 is in hexadecimal ff
```

```
255 is in hexadecimal FF
255 is in double 255.000000
255 is in double 2.550000e+002
255 is in double 2.550000E+002
255 is in compact double 255
```

Remarks

If *precision* is specified as 0 and the value to be converted is 0, the result is no characters output, as shown below:

```
Print sprintf("%.0i", 0) // no output
```

sprintf is C-compatible function. The GFA-BASIC 32 functions **Format**(), **Dec**() are easier to use.

See Also

Format, Hex$(), Oct$(), Dec$(), Using

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Using Function

Purpose

Formats an expression according to instructions contained in a format expression.

Syntax

$ = **Using**[**$**](format$**,** a)

format$:sex
a:aexp or sexp

Description

Using is the third function available to format an expression given a format-template. The others are **Format**() and **sprintf**(). **Using** is often used together with **Print**; in older BASICs Using was exclusively reserved for Print. In GFA-BASIC 32 **Using** is a separate function and can be used in a **Print** expression as in the old days.

```
f$ = Using("###.##", 2.1)
```

```
Print Using("###.##", 2.1)
```

The following characters are available for formatting of numerical expressions:

#	Place holder for a digit. When this digit is the last digit in the format template it is rounded off before output. This is used to indicate the decimal point in between the # characters.

.	Breaks decimal numbers in several # characters.
,	Inserts a comma at the corresponding place between the # characters and can, for example, be used to separate the thousands.
-	Reserves a place for the minus sign. If the number is positive a space is printed instead. This format character is only allowed before or after the formatting template.
+	Similar to the - character only a plus sign is displayed before of after a positive number. The plus and the minus characters cannot be combined.
*****	An alternative to #, the leading zeros are replaced by spaces.
$	When placed immediately before the very first #, it performs the printing of a $ sign in front of the number.
^	Sets the exponential format (E+000). In this format the # character specifies the length of the mantissa, while the ^ character specifies the length of the exponents including the E+ or E-. If there are several # characters before the decimal point, the exponent is adjusted so that it's divisible by the count of these characters. The negative numbers must contain the sign character.

The following characters are available for formatting of string expressions:

&	Performs the output of the whole string.
!	Limits the output to the first character in the string.
\..	Specifies the number of characters to be printed from a string. The count includes both \ characters.
_	An underline performs the output of the next character in template as a literal.

Example

```
OpenW 1
Local a%, f1$, f2$, f3$, f4$
f1$ = "#,###"
f2$ = "#,###_._._."
f3$ = "\...\"
f4$ = "###.###^^^^"
//
Print Using(f1$, PI)//  prints 3.142
Print Using(f2$, PI)//  prints 3.142...
Print Using(f3$, "Hallo GFA")//  prints Hallo
Print Using(f4$, 2 ^ 10)//  prints 1.024E+03
```

Remarks

The decimal point and comma can be swapped using the **Mode Using** ".," or ",." command.

See Also

Str(), Print, Format, sprintf, Mode

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

CByte, CBool, CCur, CDate, CDbl, CShort, CInt, CLong, CHandle, CLarge, CSng, CFloat, CStr, CVar Functions

Purpose

Each function coerces an expression to a specific data type.

Syntax

Bool = **CBool**(expression)

Byte = **CByte**(expression)

Currency = **CCur**(expression)

Date = **CDate**(expression)

Double = **CDbl**(expression)

Short = **CShort**(expression)

Integer = **CInt**(expression)

Long = **CLong**(expression)

Handle = **CHandle**(expression)

Large = **CLarge**(expression)

Single = **CSng**(expression)

Single = **CFloat**(expression)

String = **CStr**(expression)

Variant = **CVar**(expression)

expression : string expression or numeric expression

Description

If the expression passed to the function is outside the range of the data type being converted to, an error occurs.

In general, you can document your code using the data-type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CCur** to force currency arithmetic in cases where single-precision, double-precision, or integer arithmetic normally would occur.

You should use the data-type conversion functions instead of **Val** to provide internationally aware conversions from one data type to another. For example, when you use **CCur**, different decimal separators, different thousand separators, and various currency options are properly recognized depending on the locale setting of your computer.

Use the **IsDate** function to determine if *date* can be converted to a date or time. **CDate** recognizes date literals and time literals as well as some numbers that fall within the range of acceptable dates. When converting a number to a date, the whole number portion is converted to a date. Any fractional part of the number is converted to a time of day, starting at midnight.

CDate recognizes date formats according to the locale setting of your system. The correct order of day, month, and year may not be determined if it is provided in a format other than one of the recognized date settings. In addition, a long date format is not recognized if it also contains the day-of-the-week string.

The base year, at least for Windows 98, is 1930. This means that years specified with only two digits are interpreted a based to 1930. A year of "29" means 2029, and "31" means 1931. Since this is OLE dependent, located in oleaut32.dll, it cannot be adjusted.

! The integer type conversion functions always round to the nearest even number! When the fractional part is exactly 0.5, **CByte**, **CShort**, **CInt, CLong, and CLarge** always round it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2.

```
Print CInt(1.5)    // 2
Print CByte(0.5)   // 0
```

CByte, **CShort**, **CInt, CLong, and CLarge** differ from the **Fix** and **Int** functions, which truncate, rather than round, the fractional part of a number. Also, **Fix** and **Int** always return a value of the same type as is passed in.

Note **CByte** is the only function that returns an unsigned value (0 .. 256). A negative parameter is converted to a positive value.

CStr returns a string depending on the type of the argument passed:

Boolean	0 or -1
Number	string containing the number

Date	short date format
Empty	zero-length string ("")
Null	run-time error
Array	byte copy to string, see CStr(a()).

Example

```
Print CBool(25 < 24 < 30) //result: True
Print CBool(25 > 24 > 30) //result: False
Print CByte(0.49999)      //result: 0
Print CByte(0.50001)      //result: 1
Print CByte(-1.6)         //result: 254
Dim v                     '= Null
Print CStr(12.2)          //result: 12.2
Print CStr(1 > 0)         //result: -1
Print CStr(v)             //result:
Print CStr(Date)          //result: today's date
```

Remarks

GFABasic does not support the Visual Basic functions **CDec** and **CVErr** (see this page for a workaround for the latter).

Integer conversions use the Gauss rule that if you are in an perfect half case, you must round to the nearest digit that can be divided by 2 (0,2,4,6,and 8). This rule is important to obtain more accurate results with rounded numbers after operation.

An example:

Value	**Standard rounding**	**"Gaussian" rounding**
54.1754	54.18	54.18
343.2050	343.21	343.20

106.2038	106.20	106.20
Sum503.5842	503.59	503.58

The "Gaussian" sum is nearer to the unrounded sum (difference of 0.0042 with Gaussian and 0.0058 with Standard rounding.)

Another example with half-round cases only:

Unrounded	**Standard rounding**	**"Gaussian rounding"**
27.25	27.3	27.2
27.45	27.5	27.4
27.55	27.6	27.6
Sum82.25	82.4	82.2

Again, the "Gaussian" rounding result is nearer from the unrounded result than the "Standard" one.

See Also

Fix(), Int(), Round(), CByteRZ(), CIntRZ(), CLargeRZ(), CLongRZ(), CShortRZ()

{Created by Sjouke Hamstra; Last updated: 05/04/2018 by James Gaite}

Fix, Int, Floor & Ceil and Trunc & Frac Functions

Purpose

Return the integer or fractional portion of a numeric expression.

Syntax

n = **Ceil**(x)
n = **Floor**(x)

n = **Fix**(x)
n = **Int**(x)

n = **Trunc**(x)
f = **Frac**(x)

f	*: floating point variable*
x	*: any numeric variable*
n	*: integer*

Description

Ceil rounds up *x* to the next largest integer, while **Floor** rounds it down to the next smallest integer.

Trunc removes the fractional element of *x* and returns the integer, while **Frac** does the opposite and returns the fraction.

Finally, **Int** acts like **Floor** and rounds *x* down, while **Fix** is synonymous with **Trunc** and simply returns its integer element.

Example

```
Debug.Show
Debug "-- With Positive Numbers --"
Trace Ceil(3.4)          // Output: 4
Trace Floor(3.4)         // Output: 3
Trace Trunc(3.4)         // Output: 3
Trace Frac(3.4)          // Output: 0.4
Trace Int(3.4)           // Output: 3
Trace Fix(3.4)           // Output: 3
Debug
Debug "-- With Positive Numbers --"
Trace Ceil(-3.4)         // Output: -3
Trace Floor(-3.4)        // Output: -4
Trace Trunc(-3.4)        // Output: -3
Trace Frac(-3.4)         // Output: -0.4
Trace Int(-3.4)          // Output: -4
Trace Fix(-3.4)          // Output: -3
```

Remarks

CInt, or any of the integer Cxxx functions, acts differently to **Int** as it rounds the passed number to the nearest integer, as does **Round**.

See Also

CInt, FRound(), QRound(), Round()

{Created by Sjouke Hamstra; Last updated: 27/01/2016 by James Gaite}

CByteRZ, CShortRZ, CIntRZ, CLongRZ, CLargeRZ Functions

Purpose

Each function coerces an expression to a specific integer data type rounding towards zero.

Syntax

Byte = **CByteRZ**(expression)

Short = **CShortRZ**(expression)

Integer = **CIntRZ**(expression)

Long = **CLongRZ**(expression)

Large = **CLargeRZ**(expression)

expression: aexp

Description

Converts a numeric or string expression to a specific integer data type rounding towards zero (RZ).

When *expression* is a string the value in the string is converted to a value using the regional settings. You should use the data-type conversion functions instead of **Val** to provide internationally aware conversions from one data type to another. For example, when you use **CLongRZ**, different decimal separators, different thousand separators, and various currency options are properly recognized depending on the locale setting of your computer.

In general, you can document your code using the data-type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type.

Note **CByteRZ** is the only function that returns an unsigned value (0 .. 256). A negative parameter is converted to a positive value.

Example

```
Print CByteRZ(1.999) //result: 1
Print CByteRZ("2,1") //result: 21 (UK or USA) or 2
  (European)
Print CByteRZ(-1.6)  //result: 255
```

See Also

CByte, CShort, CInt, CLong, CLarge

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Asc Function

Purpose

Determines the ASCII value of a character in a string.

Syntax

% = **Asc**(a$ [, offset = 1])

a$:sexp

offset:numeric expression

Description

Asc(a$) returns the ASCII code of the first character in a$. If a$ is blank a 0 is returned. **Asc**(a$, n) returns the ASCII code of the n-the character in a$.

Example

```
OpenW # 1
Print Asc("TEST")    //prints 84 since 84 is the
 ASCII code for T.
Print Asc("TEST", 2) //prints 69 since 69 is the
 ASCII code for E.
```

See Also

Mid$()

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Val Functions

Purpose

converts a string expression into a number.

Syntax

# = **Val**(value) - *Floating point conversion*

# = **ValDbl**(value) - *Floating point conversion*

cvar = **ValCur**(value) - *Currency conversion*

dvar = **ValD**(value) - *Floating point and/or Date conversion*

dvar = **ValDate**(value) - *Floating point and/or Date conversion*

% = **ValInt**(value) - *Integer (32 Bit)*

Large = **ValLarge**(value) - *Integer (64 Bit)*

value:sexp or Date

Description

These functions convert a string or a Date in a numeric value. The type of the return value depends on the Val function used.

Val() and **ValDbl**() return a floating point value of type Double. **ValCur**() returns a Currency-value. **ValD**() and **ValDate**() return a Date-value (note that the date must be in dd.mm.yy[yy] rather than dd/mm/yyyy format), and

ValInt and **ValLarge** return 32-bit and 64-bit integers respectively.

If during conversion **Val**() encounters a character which cannot be interpreted as a part of a number ("1234a" for example), the evaluation of the string expression is terminated and the number obtained up until this point (1234 in the above example) is then returned; if the string expression begins with a character which cannot be interpreted as a part of a number, **Val**() returns 0. The Val?() function can be used to discover how many characters will be converted and, thus, whether all the characters or just some are eligible.

The **Val** function recognizes the period (.) as a valid decimal separator. However this can be influenced by setting the decimal separator with **Mode Val**. Using **Mode Val** the comma can be used as a decimal separator as well.

If the string expression begins with &X or %, then binary conversion takes place. &O or &Q converts to octal, while &H, & or $ converts to hexadecimal.

Mode BaseYear *sexp* sets the year used as base for dates enetered with **ValD** and **ValDate**. The default (1930) defines annual numbers between "30" and "99" and the values are interpreted as being from 1930 to 1999. The values between "00" and "29" are according to the years 2000 to 2029.

Example

```
Debug.Show
Trace Val("-.123")       // Prints -0.123
Local a$ = Str$(12345) : Trace a$
Trace Val(a$)            // Prints 12345
Trace Val("&H" + "AF")   // Prints 175
Trace Val("$AA")         // Prints 170
Trace Val("%10101011")   // Prints 171
Trace ValD("16.09.15")   // Prints 12.10.2015
Trace ValD("16/09/15")   // Prints 00:00:00 (Only
  works with German date format)
```

For examples on using **Mode BaseYear** and **Mode Val** see here.

Remarks

The **Val** and **ValDbl** functions don't convert string to numeric values according the regional settings. Instead, **Val** and **ValDbl** use the internal GFA-BASIC 32 **Mode Val** setting. To make sure a program acts according the regional settings use **CDbl**().

See Also

CDbl, Val?

{Created by Sjouke Hamstra; Last updated: 16/09/2015 by James Gaite}

Val? Function

Purpose

Determines the size of a string expression containing a number when using **Val**().

Syntax

% = **Val?**(a$)

Description

Val?(a$) returns 0, if a$ contains no characters that can be interpreted as numbers.

Example

```
Debug.Show
Trace Val?("12345")      // Prints 5
Trace Val?("3.00 DM")    // Prints 4
Trace Val?("Hallo GFA") // Prints 0
```

See Also

Val, Format, CDbl

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

CheckSumLong Function

Purpose

Computes the checksum for a range of bytes returning a 32-bit integer.

Syntax

sum = **CheckSumLong**(addr, count, [old])

sum, addr, count, old: iexp

Description

The function **CheckSumLong**() calculates a simple checksum (Long value) for a block of data: *count* bytes from the address *addr*. The optional parameter *old* is to be used if you want to create a checksum for more than one block, *old* must contain the checksum for the other block.

The checksum is a simple adding of 32-bit values in the data.

Example

```
Local a$, b$, ch_a%
Debug.Show
a$ = "This is a test"
b$ = "another block"
ch_a% = CheckSumLong(V:a$, Len(a$))
Trace CheckSumLong(V:a$, Len(a$))          //
  -112105912
```

```
Trace CheckSumLong(V:b$, Len(b$))          //
  -128892778
Trace CheckSumLong(V:a$, Len(a$), ch_a%)   //
  -224211823
```

Remarks

The calculation of data with **CheckSumByte**, **CheckSumShort**, **CheckSumLong** (or **CheckXorxxx**()) is very fast (up to 10 times faster than **Crc16**() or **Crc32**()).

A checksum is a form of redundancy check, a simple way to protect the integrity of data by detecting errors in data that are sent through space (telecommunications) or time (storage). It works by adding up the basic components of the data, typically the asserted bits, and storing the resulting value. Anyone can later perform the same operation on the data, compare the result to the authentic checksum, and (assuming that the sums match) conclude that the data was probably not corrupted.

See Also

CheckSumByte(), CheckSumLong(), CheckSumShort(), CheckXorByte(), CheckXorLong(), CheckXorShort(), Crc16(), Crc32()

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

CheckSumShort Function

Purpose

computes the checksum for a range of bytes returning a 16-bit integer.

Syntax

w = **CheckSumShort**(addr, count, [old])

w, old:16-bit integer
addr, count:iexp

Description

The function **CheckSumShort**() calculates a simple checksum (16-bit integer value) for a block of data: *count* bytes from the address *addr*. The optional parameter *old* is to be used if you want to create a checksum for more than one block, *old* must contain the checksum for the other block.

The checksum is a simple adding of 16-bit values in the data.

Example

```
Local a$, b$, ch_a&
Debug.Show
a$ = "This is a test"
b$ = "another block"
ch_a& = CheckSumShort(V:a$, Len(a$))
```

```
Trace CheckSumShort(V:a$, Len(a$))          //
  24474
Trace CheckSumShort(V:b$, Len(b$))          //
  14568
Trace CheckSumShort(V:a$, Len(a$), ch_a&)   //
  -16588
```

Remarks

The calculation of data with **CheckSumByte**, **CheckSumShort**, **CheckSumLong** (or **CheckXorxxx**()) is very fast (up to 10 times faster than **Crc16**() or **Crc32**()).

A checksum is a form of redundancy check, a simple way to protect the integrity of data by detecting errors in data that are sent through space (telecommunications) or time (storage). It works by adding up the basic components of the data, typically the asserted bits, and storing the resulting value. Anyone can later perform the same operation on the data, compare the result to the authentic checksum, and (assuming that the sums match) conclude that the data was probably not corrupted.

See Also

CheckSumByte(), CheckSumLong(), CheckSumShort(), CheckXorByte(), CheckXorLong(), CheckXorShort(), Crc16(), Crc32()

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

CheckXorLong Function

Purpose

Computes the checksum for a range of bytes returning a 32-bit value.

Syntax

l = **CheckXorLong**(addr, count, [old])

l, addr, count, old:iexp

Description

The function **CheckXorLong**() calculates a simple checksum (Long value) for a block of data: *count* bytes from the address *addr*. The optional parameter *old* is to be used if you want to create a checksum for more than one block, *old* must contain the checksum for the other block.

The checksum is a simple XOR-ing of 32-bit values (4 bytes) in the data.

Example

```
Local a$ = "This is a Test"
Print CheckXorLong(V:a$, Len(a$)) // 911103334
Dim a#(10), b#(10)
Mat Set a#() = 120
Mat Set b#() = -234
Dim cha_xor% = CheckXorLong(V:a#(0), ArraySize(a#
  ()))
```

```
Dim ch_xor% = CheckXorLong(V:b#(0), ArraySize(b#
  ()), cha_xor%)
Print cha_xor%, ch_xor% // 1079902208, -2144124928
```

Remarks

The calculation of data with **CheckXorByte**, **CheckXorShort**, **CheckXorLong** (or **CheckSumxxx**()) is very fast (up to 10 times faster than **Crc16**() or **Crc32**()).

A checksum is a form of redundancy check, a simple way to protect the integrity of data by detecting errors in data that are sent through space (telecommunications) or time (storage). It works by adding up the basic components of the data, typically the asserted bits, and storing the resulting value. Anyone can later perform the same operation on the data, compare the result to the authentic checksum, and (assuming that the sums match) conclude that the data was probably not corrupted.

See Also

CheckSumByte(), CheckSumLong(), CheckSumShort(), CheckXorByte(), CheckXorLong(), CheckXorShort(), Crc16(), Crc32()

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

CheckXorShort Function

Purpose

Computes the checksum for a range of bytes returning a 16-bit value.

Syntax

w = **CheckXorShort**(addr, count, [old])

w, old:16-bit integer
addr, count:iexp

Description

The function **CheckXorLong**() calculates a simple checksum (Long value) for a block of data: *count* bytes from the address *addr*. The optional parameter *old* is to be used if you want to create a checksum for more than one block, *old* must contain the checksum for the other block.

The checksum is a simple XOR-ing of 16-bit (2 bytes) values in the data.

Example

```
Local a$ = "This is a Test"
Print CheckXorShort(V:a$, Len(a$)) // 25384
Dim a#(10), b#(10)
Mat Set a#() = 120
Mat Set b#() = -234
Dim cha_xor& = CheckXorShort(V:a#(0), ArraySize(a#
  ()))
```

```
Dim ch_xor& = CheckXorShort(V:b#(0), ArraySize(b#
  ()), cha_xor&)
Print cha_xor&, ch_xor& // 16478, -16333
```

Remarks

The calculation of data with **CheckXorByte**, **CheckXorShort**, **CheckXorLong** (or **CheckSumxxx**()) is very fast (up to 10 times faster than **Crc16**() or **Crc32**()).

A checksum is a form of redundancy check, a simple way to protect the integrity of data by detecting errors in data that are sent through space (telecommunications) or time (storage). It works by adding up the basic components of the data, typically the asserted bits, and storing the resulting value. Anyone can later perform the same operation on the data, compare the result to the authentic checksum, and (assuming that the sums match) conclude that the data was probably not corrupted.

See Also

CheckSumByte(), CheckSumLong(), CheckSumShort(), CheckXorByte(), CheckXorLong(), CheckXorShort(), Crc16(), Crc32()

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

PackMem, UnPackMem Function

Purpose

Compresses a block of memory into a string.

Syntax

$ = PackMem(address, length [,flag = 0])

$ = UnPackMem(string, length)

address, length: iexp
string:sexp
flag:iexp

Description

The function **PackMem** returns a compressed string from a block of memory at the specified address and length. The function **UnPackMem** decompresses a string compressed with **PackMem**.

PackMem will place a 12 byte label in front of a compressed string. The first four signs are "PCK0" (PeCehKahZero), after this, four more signs follow with the length of the compressed data and last four with the original length:

"PCK0" + Mkl$(length_after_compression) + Mkl$(original length) + packed data

When both the original data size as the compressed data size are smaller than 65536, a header of 8 bytes is used, with a lowercase k instead of K, and both lengths in a 16-bit value. Data that cannot be compressed (random byte sequences or a Crypt$) are marked with a lowercase c, followed by only one length (k=16 bit, K=32 bit), so 6 or 8 bytes.

The optional *flag* can have a value of 0, 1, or 2. If flag = 1 an additional bit pack run is performed. This run will take a bit of time, but as a result, you get a better compression rate (1-10%, sometimes more). In addition, plain text snippets are mostly removed from the compressed string. Packing with default value of flag (= 0) often results in a compressed string where words might be readable. A packed string with flag is 1 is marked as PCK1 or PCk1 instead of PCK0.

flag = 2 forces a bit pack, whether or not the packed string becomes longer.

Example

```
OpenW 1
Local a$, b$, c$, d$, e%, x%, b1$
// read
a$ = Peek$(4096 * 1024, 60000)
// pack the first part into b$, and
// the rest into c$
b$ = PackMem(V:a$, 30000)
c$ = PackMem(V:a$ + 30000, 30000)
// unpack
d$ = UnPackMem(V:b$, Len(b$)) + UnPackMem(V:c$,
  Len(c$))
// display: before, packed 2 x times, after
Print Len(a$), Len(b$), Len(c$), Len(d$)
```

```
// comparison: before - after
Print a$ = d$
// all with flag 1
b$ = PackMem(V:a$, 30000, 1)
c$ = PackMem(V:a$ + 30000, 30000, 1)
d$ = UnPackMem(V:b$, Len(b$)) + _
  UnPackMem(V:c$, Len(c$))
Print Len(a$), Len(b$), Len(c$), Len(d$)
Print a$ = d$
//all with flag 2
b$ = PackMem(V:a$, 30000, 2)
c$ = PackMem(V:a$ + 30000, 30000, 2)
d$ = UnPackMem(V:b$, Len(b$)) + _
  UnPackMem(V:c$, Len(c$))
Print Len(a$), Len(b$), Len(c$), Len(d$)
Print a$ = d$
```

Remarks

The compression rate of **PackMem** compares to ARC, the grand father of all compression programs, or Compress the program from Microsoft.

See Also

Pack$

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

UUEncode and UUDecode Functions

Purpose

Encodes and decodes a string using UUE encoding.

Syntax

uustring$ = **UUEncode**(string$)

string$ = **UUDecode**(uustring$)

Description

UUEncode converts a string to the UUE format. This a relative old format used for mailboxes, where special characters are being replaced by printable characters.

UUDecode decodes the UUE encoded string.

Example

```
OpenW 1
Local a$, s_mime$, x%, s$
a$ = "GFA Software Technologies GmbH"
s_mime$ = uuencode(a$)
Print s_mime$
s$ = uudecode(s_mime$)
Print s$
```

Remarks

See Also

MemToMiMe(), MemToUU(), MiMeToMem(), MiMeDecode(), MiMeEncode(), UUToMem()

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Is Operator

Purpose

Used to compare two object reference variables

Syntax

Bool **=** object1 **Is** object2

Bool = TypeOf(object1) **Is** *typename*

Description

If object1 and object2 both refer to the same object, result is True; if they do not, result is False. Two variables can be made to refer to the same object in several ways.

In the following example, A has been set to refer to the same object as B:

Set A = B

The **Is** operator is an object reference comparison operator. It does not compare objects or their values; it checks only to determine if two object references refer to the same object.

The **Is** operator is also used together with **TypeOf**. In this case **Is** compares two OCX or OLE types.

Example

```
OpenW 1
Ocx TextBox tb1
Ocx TextBox tb2
Ocx Command bt1
Ocx Command bt2
Ocx Command bt3
Print tb1 Is tb2                    // False
Set tb1 = tb2
Print tb1 Is tb2                    // True
Set bt1 = bt3
Print bt2 Is bt1                    // False
Print TypeOf(bt2) Is Command        //True
```

Remarks

When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last. The **Is** operator is evaluated last.

See Also

Set, TypeOf

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

TypeOf Function

Purpose

Queries the type of an object

Syntax

If TypeOf(object) **Is** objecttype

If TypeOf object **Is** objecttype

object:OLE object
objectname:OLE type name

Description

TypeOf is always part of an **If** expression, of the form **TypeOf** *objectname* **Is** *objecttype*. The *object* is any object reference and *objecttype* is any valid object type. The expression is **True** if *objectname* is of the object type specified by *objecttype*; otherwise it is False.

Example

```
OpenW 1
Local obj As Object
Ocx Command cmd1
Set obj = cmd1 : result(obj)
Set obj = Win_1   : result(obj)
Set obj = Nothing   : result(obj)

Function result(obj As Object)
  Try
```

```
    If TypeOf(obj) Is Command
      Print obj.name & " is a Command Button"
    Else
      Print obj.name & " is not a Command Button. It
        is a " & TypeName(obj) & "."
    EndIf
  Catch
    Print "The Object is set to Nothing"
  EndCatch
EndFunc
```

Remarks

Select Case may be more useful when evaluating a single expression that has several possible actions. However, the **TypeOf** *objectname* **Is** *objecttype* clause can't be used with the **Select Case** statement.

TypeOf can only be used with Objects but, as seen in the example above, does not recognise when the object is set to Nothing and returns an error. Therefore, it should always be contained within a Try/Catch construct if there is even the remotest possibility of the object being queried not having been defined and this is especially true if you are querying the edit box of a ComboBox which returns Nothing eventhough it has been defined. **TypeName** could be used instead as it recognises both Objects and the Nothing state of undefined objects. This is shown best by the following example:

```
Type COMBOBOXINFO
  - Long cbsize
  rcItem As RECT
  rcButton As RECT
  - Long stateButton, hwndCombo, hwndItem, hwndList
EndType
Type RECT
```

```
  - Long Left, Top, Right, Bottom
EndType
Global Const CB_GETCOMBOBOXINFO = 0x0164
Ocx ComboBox cb = "", 10, 10, 100, 22
Local a$, ci As COMBOBOXINFO : ci.cbsize =
  SizeOf(COMBOBOXINFO)
~SendMessage(cb.hWnd, CB_GETCOMBOBOXINFO, 0, V:ci)
Text 10, 50, "ComboBox Edit BoxHandle: " &
  ci.hwndItem
Try
  If TypeOf(cb) Is ComboBox Then a$ = a$ & "TypeOf
    recognises the ComboBox"
  If TypeOf(OCX(ci.hwndItem)) Is TextBox Then a$ =
    a$ & " and the Edit Box"
Catch
  a$ = a$ & " but not the Edit Box as it is not an
    OCX object and returns Nothing, "
  a$ = a$ & "as is shown by
    TypeName(OCX(ci.hwndItem)) = " & #34 &
    "Nothing" & #34 & " being returned as" &
    (TypeName(OCX(ci.hwndItem)) = "Nothing")
EndCatch
Text 10, 65, a$
Do : Sleep : Until Me Is Nothing
```

See Also

TypeName, VarType

{Created by Sjouke Hamstra; Last updated: 14/09/2015 by James Gaite}

Now, Now$ Function

Purpose

Returns a **Date** specifying the current date and time according your computer's system date and time.

Syntax

d = **Now**

$ = **Now$**[(d)]

d: Date expression

Description

Now returns a **Date** specifying the current date and time according your computer's system date and time.

Now$** returns the current date and time as a string formatted according Mode Date setting (dd.mm.yyyy HH:mm:ss). **Now$(date) returns the specified Date as a string in the same format.

Example

```
Dim MyDate As Date
MyDate = Now    ' MyDate contains the current
  system date/time.
Dim s$ = Now$
Print "Now = "; MyDate
Print "Now$ = "; s$
```

Remarks

Now$**[()] is identical to **DateTime$[()]

See Also

DateTime$, Date, Date$, Time, Time$

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

_time Function

Purpose

Gets the system time

Syntax

int = **_time** [(V: x%)]

x : ivar

Description

The **_time** function returns the number of seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time, according to the system clock. The return value is stored in the location given by x%. This parameter may be zero, in which case the return value is not stored.

Example

```
Dim a%
Print _time(V:a%)   // Number of seconds elapsed
Print a%            // - ditto -
Print _time(0)      // - ditto -
Print _time         // - ditto -
Print _ctime(0)     // Date in C Format
Print _ctime(V:a%) // - ditto -
Print _ctime        // - ditto -
```

Remarks

_**ctime** and _**time** are implemented for compatibility reasons with C. These functions are restricted to dates between 1970 and 2038 and will result in the 2K38 bug….

_**time** is the same as ((**Now** + #1.1.1970#) *24*60*60).

See Also

_ctime, Now

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Date$ Function

Purpose

Returns string for a given date or the system date.

Syntax

Date$[(date)]

date: Date expression

Description

Date$ returns the system date in the following format:

DD.MM.YYYY (Day.Month.Year)

MM.DD.YYYY (Month.Day.Year; US format)

YYYY-MM-TT (Year-Month-Day, international format)

The format is set with the **Mode** command.

Example

```
OpenW # 1
Print Date$
// Change to US Date mode
// Mode Format does not work with Date$
Mode Date "/"
Print Date$( Date - 30 )
// For UK Date style use...
Print Format(Date, "dd/mm/yyyy")
```

Remarks

There is no command to set the system time, because setting the time requires the SE_SYSTEMTIME_NAME privilege.

Time$() returns the time part of a date. **DateTime**$() returns both the date and the time.

Use **Format** to convert a Date to a different format.

See Also

Format, Time$, DateTime$, Date, Time, Now, Mode

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

Time$ Function

Purpose

Returns a string for the specified time.

Syntax

$ = **Time$**[(date])

date:optional, date exp

Description

Time$ returns the specified time, or when not used the system time, in the following format: HH:MM:SS (Hours:Minutes:Seconds)

Example

```
OpenW # 1
Print Time$
Local x As Date = #12.12.2001 18:42:16#
Print Time$(x)
' a simple calculation
Print Time$(x + #03:00:00#)
```

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

DateTime$ Function

Purpose

Converts a Date to a string.

Syntax

$ = **DateTime$**(date)

date: Date expression

Description

Together with **Date$**() and **Time$**() the function **Now$** and **DateTime$**() converts a date to a printable string. The output format is set with **Mode Date** and **Mode Time**. **DateTime$** includes both the date and the time part.

Example

```
Debug.Show
Trace DateTime$(Now)
Trace Now$
// Prints the actual date and time
Trace DateTime$(Date)
// Prints only the actual date
Trace DateTime$(11111.1111)
// Prints 06/02/1930 02:34:59
Local d As Date = 31344.55
Trace d
Trace DateTime$(d / 4 - 2 * 3)
// Prints 06/08/1921 03:18:00
Trace DateTime$(d)
```

```
// Prints 10/24/1985 13:12:00
```

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

Day Function

Purpose

Returns a whole number between 1 and 31, inclusive, representing the day of the month.

Syntax

Day(*time*)

Description

The *time* argument is any expression that can represent a time.

Example

```
Debug.Show
Local z As Date = HmsToTime(110000, 20, 4000)
Trace Day(z)
Trace Day(Now)
Trace Day(Date)
Trace Day(#12:12:12#)
Trace FileDateTime("c:\windows\notepad.exe")
Trace Day(FileDateTime("c:\windows\notepad.exe"))
```

Remarks

You can indicate a time in hours, minutes and seconds, separated by " : "; or only with four numbers for hours and minutes (with or without the using of AM or PM). Using only 4 characters forces the GFABASIC 32 editor to automatically add ":00" for the seconds. When using AM or PM the editor automatically converts to 24 hour mode. For instance, "#2:24#" will automatically convert to #14:24:00#, and (#2:24AM#) => (#02:24:00#).

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

DayNo Function

Purpose

Returns a whole number between 1 and 365 (366 for leap years), inclusive, representing the day of the year.

Syntax

DayNo(*time*)

Description

The *time* argument is any expression that can represent a time.

Example

```
Debug.Show
Local z = HmsToTime(110000, 20, 4000)
Trace z
Trace DayNo(z)
Trace DayNo(Now)
Trace DayNo(Date)
Trace DayNo(#12:12:12#)
Trace
  DayNo(FileDateTime("c:\windows\notepad.exe"))
```

Remarks

You can indicate a time in hours, minutes and seconds, separated by " : "; or only with four numbers for hours and minutes (with or without the using of AM or PM). Using only 4 characters forces the GFABASIC 32 editor to automatically add ":00" for the seconds. When using AM or PM the editor automatically converts to 24 hour mode. For instance, "#2:24#" will automatically convert to #14:24:00#, and (#2:24AM#) => (#02:24:00#).

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

Month Function

Purpose

Returns an Integer specifying a whole number between 1 and 12, inclusive, representing the month of the year.

Syntax

Month(*date***)**

date:Date exp

Description

The function **Month**() retunrs the month of a Date.

Example

```
OpenW 1
Local z As Date
z = HmsToTime(110000, 20, 4000)
Print z, Month(z)
Print Now, Month(Now)
Print Date, " ", Month(Date)
Print "12/12/1912", " ", Month(#12.12.1912#)
Print FileDateTime("c:\windows\notepad.exe"),
 Month(FileDateTime("c:\windows\notepad.exe"))
```

Remarks

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

WeekDay Function

Purpose

Returns an Integer specifying a whole number (1-6) containing the day of the week,

Syntax

WeekDay(*date***)**

date:Date exp

Description

The function **WeekDay**() returns the day of the week, relative to Sunday.

Sunday = 1, Monday = 2, Tuesday = 3, Wednesday = 4, Thursday = 5, Friday = 6, Saturday = 7

The **Mode Lang** command determines use of the regional settings.

Example

```
OpenW 1
Local z As Date
z = HmsToTime(110000, 20, 4000)
Print z, WeekDay(z), Format(WeekDay(z), "dddd")
Print Now, WeekDay(Now), Format(WeekDay(Now),
  "dddd")
Print Date, " ", WeekDay(Date),
  Format(WeekDay(Date), "dddd")
```

```
Print "12/12/1912", " ", WeekDay(#12.12.1912#),
  Format(WeekDay(#12.12.1912#), "dddd")
Print FileDateTime("c:\windows\notepad.exe"),
  WeekDay(FileDateTime("c:\windows\notepad.exe")),
Print
  Format(WeekDay(FileDateTime("c:\windows\notepad.e
  xe")), "dddd")
```

Remarks

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Year Function

Purpose

Returns an Integer specifying a whole number representing the year.

Syntax

Year(*date***)**

date:Date exp

Description

The function **Year**() returns the year a Date. The result is a two digit or a four digit number, depending on the **Mode Lang** setting.

Example

```
OpenW 1
Local z As Date
z = HmsToTime(110000, 20, 4000)
Print z, Year(z)
Print Now, Year(Now)
Print Date, " ", Year(Date)
Print "12/12/1912", " ", Year(#12.12.1912#)
Print FileDateTime("c:\windows\notepad.exe"),
  Year(FileDateTime("c:\windows\notepad.exe"))
```

Remarks

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

Hour Function

Purpose

Returns a whole number between 0 and 23, inclusive, representing the hour of the day.

Syntax

Hour(*time***)**

time: Date, Variant, or String

Description

The *time* argument is any expression that can represent a time. If *time* contains **Null**, **Null** is returned.

The following example uses the **Hour** function to obtain the hour from the current time:

Example

```
Debug.Show
Local z As Date, x%
z = HmsToTime(110000, 20, 4000)
Trace z
Trace Hour(z)
Trace Hour(#16:24:12#)
Trace Hour(Now)
Trace Hour(Time)
Trace Hour(FileDateTime("c:\windows\notepad.exe"))
```

Remarks

The format of the output can be changed with the using of Mode Date..., Mode Format..., Format....

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

Minute Function

Purpose

Returns a whole number between 0 and 59, inclusive, representing the minute of the hour.

Syntax

Minute(*time*)

Description

The *time* argument is any expression that can represent a time.

Example

```
OpenW 1
Local z As Date
z = HmsToTime(110000, 20, 4000)
Print z, Minute(z)
Print Now, Minute(Now)
Print Date, Minute(Date)
Print "12:12:12", Minute(#12:12:12#)
Print FileDateTime("c:\windows\notepad.exe"),
  Minute(FileDateTime("c:\windows\notepad.exe"))
```

Remarks

You can indicate a time in hours, minutes, and seconds, separated by " : "; or only with four numbers for hours and minutes (with or without the using of AM or PM). Using only 4 characters forces the GFABASIC 32 editor to automatically add “:00“ for the seconds. When using AM or PM the editor automatically converts to 24 hour mode. For instance, "#2:24#" will automatically convert to #14:24:00#, and (#2:24AM#) => (#02:24:00#).

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

Second Function

Purpose

Returns an Integer specifying a whole number between 0 and 59, inclusive, representing the second of a minute.

Syntax

Second(time**)**

time:Date exp

Description

The function **Second**() returns the second of a Date.

Example

```
OpenW 1
Local z As Date
z = HmsToTime(110000, 20, 4000)
Print z, Second(z)
Print Now, Second(Now)
Print Date, " ", Second(Date)
Print "12:12:12", " ", Second(#12:12:12#)
Print FileDateTime("c:\windows\notepad.exe"),
  Second(FileDateTime("c:\windows\notepad.exe"))
```

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(),

DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

DateAdd Function

Purpose

Returns a **Date** containing a date to which a specified time interval has been added.

Syntax

DateAdd(interval, number, date**)**

interval:sexp
number:iexp
date:any date exp

Description

You can use the **DateAdd** function to add or subtract a specified time interval from a date. For example, you can use **DateAdd** to calculate a date 30 days from today.

The *interval* argument can have the following values:

"yyyy"	Year
"q"	Quarter
"m"	Month
"y"	Day of year
"d"	Day
"w"	Weekday (1 = Sunday, … , 7 = Saturday)
"ww"	Week

The **DateAdd** function won't return an invalid date. The following example adds one month to January 31:

DateAdd("m", 1, "31-Jan-95")

In this case, **DateAdd** returns 28-Feb-95, not 31-Feb-95. If date is 31-Jan-96, it returns 29-Feb-96 because 1996 is a leap year.

If the calculated date would precede the year 100 (that is, you subtract more years than are in date), an error occurs.

Example

```
Global x As Date, a%
x = DateAdd("ww", 4, Date) : Print x
// the actual date plus 4 weeks
x = DateAdd("yyyy", -9, Date) : Print x
// the year minus 9
x = DateAdd("m", -6, #12/31/1920#) : Print x
// the given date minus 6 month
// results 30/06/1920
```

Remarks

You can indicate a time in hours, minutes and seconds, separated by " : "; or only with four numbers for hours and minutes (with or without the using of AM or PM). Using only 4 characters forces the GFABASIC 32 editor to automatically add ":00" for the seconds. When using AM or PM the editor automatically converts to 24 hour mode. For instance, "#2:24#" will automatically convert to #14:24:00#, and (#2:24AM#) => (#02:24:00#).

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(),

DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

DateDiff Function

Purpose

Returns a **Long** specifying the number of time intervals between two specified dates.

Syntax

DateDiff(interval, date1, date2**)**

interval: sexp

date1, date2: date exp

Description

You can use the **DateDiff** function to determine how many specified time intervals exist between two dates. For example, you might use **DateDiff** to calculate the number of days between two dates, or the number of weeks between today and the end of the year.

The *interval* argument can have the following values:

"yyyy"	Year
"q"	Quarter
"m"	Month
"y"	Day of year
"d"	Day
"w"	Weekday (1 = Sunday, … , 7 = Saturday)
"ww"	Week

To calculate the number of days between date1 and date2, you can use either day of year ("y") or day ("d"). When interval is weekday ("w"), **DateDiff** returns the number of weeks between the two dates. If date1 falls on a Monday, **DateDiff** counts the number of Mondays until date2. It counts date2 but not date1. If interval is Week ("ww"), however, the **DateDiff** function returns the number of calendar weeks between the two dates. It counts the number of Sundays between date1 and date2. **DateDiff** counts date2 if it falls on a Sunday; but it doesn't count date1, even if it does fall on a Sunday.

If date1 refers to a later point in time than date2, the **DateDiff** function returns a negative number.

Example

```
OpenW 1
Print DateDiff("ww", #12/31/1999#, Now)
// returns the number of full weeks from
// today till the date
Print DateDiff("m", #01/01/1800#, Date)
// returns the numbers of month till ...
Print DateDiff("w", #01/01/1800#, Date)
// returns then numbers of weekends
Print DateDiff("ww", #01/01/1800#, Date)
// return the numbers of full weeks
Do : Sleep : Until Win_1 Is Nothing
```

Remarks

You can indicate a time in hours, minutes and seconds, separated by " : "; or only with four numbers for hours and minutes (with or without the using of AM or PM). Using only 4 characters forces the GFABASIC 32 editor to automatically add ":00" for the seconds. When using AM or PM the editor automatically converts to 24 hour mode. For instance, "#2:24#" will automatically convert to #14:24:00#, and (#2:24AM#) => (#02:24:00#).

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

DatePart Function

Purpose

Returns an integer containing the specified part of a given date.

Syntax

DatePart(interval, date**)**

interval: sexp

date: date exp

Description

You can use the **DatePart** function to evaluate a date and return a specific interval of time. For example, you might use **DatePart** to calculate the day of the week or the current hour.

The *interval* argument of time you want to return can have the following values:

"yyyy"	Year
"q"	Quarter
"m"	Month
"y"	Day of year
"d"	Day
"w"	Weekday (1 = Sunday, … , 7 = Saturday)
"ww"	Week
"h"	Hour

"n"	Minute
"s"	Second

Example

```
Debug.Show
Trace DatePart("d", #03/29/1997 23:44:45#)
// Prints 29
Trace DatePart("m", #03/29/1997 23:44:45#)
// Prints 3
Trace DatePart("yyyy", #03/29/1997 23:44:45#)
// Prints 1997
Trace DatePart("y", #03/29/1997 23:44:45#)
// Prints 260
Trace DatePart("q", #03/29/1997 23:44:45#)
// Prints 1
Trace DatePart("w", #03/29/1997 23:44:45#)
// Prints 7
Trace DatePart("ww", #03/29/1997 23:44:45#)
// Prints 13
Trace DatePart("h", #03/29/1997 23:44:45#)
// Prints 23
Trace DatePart("n", #03/29/1997 23:44:45#)
// Prints 44
Trace DatePart("s", #03/29/1997 23:44:45#)
// Prints 45
// or
Trace DatePart("h", 3.5)
// Prints 12
Trace DatePart("yyyy", 3.5 + 1500)
// Prints 1904
// or
Debug
Dim d As Date = 36525.9999
Trace d
Trace DatePart("yyyy", d) // 1999
Trace DatePart("m", d) // 12
Trace DatePart("d", d) // 31
Trace DatePart("h", d) // 23
Trace DatePart("n", d) // 59
Trace DatePart("s", d) // 51
// but
Trace DatePart("yyyy", Time) // 1899 (starting
 point)
```

Remarks

You can indicate a time in hours, minutes and seconds, separated by " : "; or only with four numbers for hours and minutes (with or without the using of AM or PM). Using only 4 characters forces the GFABASIC 32 editor to automatically add ":00" for the seconds. When using AM or PM the editor automatically converts to 24 hour mode. For instance, "#2:24#" will automatically convert to #14:24:00#, and (#2:24AM#) => (#02:24:00#).

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

TimeSerial Function

Purpose

Returns a **Date** for a specified hour, minute, and second.

Syntax

dt = **TimeSerial**(hour, minute, second)

dt: Date exp
hour, minute, second: iexp

Description

To specify a time, such as 11:59:59, the range of numbers for each **TimeSerial** argument should be in the normal range for the unit; that is, 0-23 for hours and 0-59 for minutes and seconds. However, you can also specify relative times for each argument using any numeric expression that represents some number of hours, minutes, or seconds before or after a certain time. The following example uses expressions instead of absolute time numbers. The **TimeSerial** function returns a time for 15 minutes before (-15) six hours before noon (12 - 6), or 5:45:00 A.M.

```
TimeSerial(12 - 6, -15, 0)
```

When any argument exceeds the normal range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 75 minutes, it is evaluated as one hour and 15 minutes. If any single argument is outside the range -32,768 to 32,767, an error occurs. If the time specified by the three arguments causes the date to fall outside the acceptable range of dates, an error occurs.

Example

```
OpenW 1
Local a As Date, x%
a = TimeSerial(1000000, 120000, 33000)
Print a
// prints : 16.03.78 21:18:16
```

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

TimeValue Function

Purpose

Returns a value containing the time.

Syntax

var = **TimeValue**(exp)

var	*: variable*
exp	*: aexp*

Description

This function converts *exp* and returns a time value in *var*. The conversion uses the VarDateFromString API and so takes into account the Regional settings of the system rather than the current GB **Mode** setting (for a GB **Mode** compliant function, see ValDate).

The expression *exp* can be a string, date, or date literal. If there is a date element then any date literal must use a period (or full stop) separator for date (25.12.2018 12:54) regardless of Mode settings.

The value returned to *var* depends on the variable type for the return value but is **Date** by default: when *var* is a **Single** or **Double** a decimal representation of the time element is returned; if *var* is a **String**, the time is returned as a string. If *var* is an integer then 0 (upto 12:00) or 1 (after 12:00) will be returned.

Example

```
Local da As Date, db As Double, i As Int, s As
  String
da = TimeValue("25 Jan 2019 11:42") : Print da  //
  11:42:00
db = TimeValue("25 Jan 2019 11:42") : Print db  //
  0.4875
i = TimeValue("25 Jan 2019 11:42") : Print i    //
  0 - Upto 12:00
s = TimeValue("25 Jan 2019 11:42") : Print s    //
  11:42:00
Print TimeValue(#25.01.2019 11:42:00#)          //
  11:42:00
Print VarType(TimeValue(#25.01.2019 11:42:00#)) //
  7 = Date
```

This example uses the **DateTime** function to convert a string to a date.

```
Dim X As Date, Y As Double
X = TimeValue(Now)
Y = TimeValue(Now)
Print X, Y, TimeValue(Now)
Print TimeValue(FileDateTime(ProgName))
```

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 02/02/2019 by James Gaite}

DateToDmy, DateToDmyHms and TimeToHms Commands

Purpose

Returns the day, month, year and/or hour, minute, and seconds of a Date/Time value.

Syntax

DateToDmy date, day, month, year
DateToDmyHms date, day, month, year, hour, minute, second
TimeToHms date, hour, minute, second

date	*: date expression*
day, month, year, hour, minute, second	*: integer expressions*

Description

These commands are shortcuts for **Day**(), **Month**(), **Year**(), **Hour**(), **Minute**(), and **Second**() functions and assign the specified part of a Date/Time value to a pre-defined integer variable. See the example below for how to use them.

Example

```
OpenW 1
Global Int d, m, y, h, mn, s
DateToDmy Date - 25, d, m, y            // Returns
  the date 25 days ago
```

```
PrintResult(1)
DateToDmyHms Now, d, m , y, h, mn, s     // Returns
  the date and time now
PrintResult(3)
TimeToHms Now - (1 / 24), h, mn, s       // Returns
  the time one hour ago
PrintResult(2)
Do : Sleep : Until Me Is Nothing

Sub PrintResult(part%)
  If Btst(part%, 0) Then Print "Day: "; d, "Month:
    "; m, "Year: "; y;
  If Btst(part%, 1) Then Print Tab(45); "Hour: ";
    h, "Minute: "; mn, "Second: "; s
  If Not Btst(part%, 1) Then Print
EndSub
```

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 13/02/2016 by James Gaite}

_TimerCount and _TimerFreq Functions

Purpose

Combined, they return the time since Windows started and form the basis of the **Timer** function.

Syntax

x = **_TimerCount**

x = **_TimerFreq**

x : i64var

Description

_TimerFreq returns the frequency with which the timer is counted (1/1193190 - resolution, better as microseconds).

_TimerCount returns the number of counts since Windows began, the rate of the counts being determined by the frequency stored in **_TimerFreq**: hence, the value of **Tmer** - or the seconds elapsed since Windows began - is theoretically **_TimerCount** / **_TimerFreq**.

Example

```
OpenW 1
Dim a, b As Large, c As Double
a = _TimerCount
b = _TimerFreq
c = Timer
```

```
Print a       // Result
Print b
Print Round(a / b, 5), Round(c, 5)  // equal to
  Timer
Do
  Sleep
Until Me Is Nothing
```

Remarks

When **Timer** is compared to (**_TimerCount** / **_TimerFreq)**, the values aren't identical. This isn't a flaw in GFA-BASIC 32, but is due to fact that both functions aren't invoked at the same time.

See Also

Timer, oTimer, qTimer

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Timer, oTimer and qTimer Functions

Purpose

Return a value of the time elapsed since Windows started in varying time intervals.

Syntax

# = **Timer**

% = **oTimer**

# = **qTimer**

Description

Timer returns the time as a Double in seconds. The internal resolution is **_TimerFreq**.

oTimer is compatible to GFA-BASIC 16 **Timer** and returns the time since the Windows start in milliseconds as a 32-bit integer value with a resolution of this timer of 1 millisecond. **oTimer** is slower than **Timer** and the resolution of **oTimer** (milliseconds as an Integer value) is also smaller than that of **Timer** (1/1.2 million (_TimerFreq)).

Finally, **qTimer** returns the time as a Double like **Timer** but has a frequency of approximately 55 milliseconds (the 18.2hz of the Timer Interrupt); **qTimer** corresponds to the VB function Timer.

Example

```
FullW 1 : AutoRedraw = 1
Global f1# = .4, f2# = .85, _
  w1# = 35, w2# = 5
f1 = .4, f2 = .85
Color RGB(0, 255, 0)
Local t# = Timer, qt# = qTimer, ot% = oTimer
xdraw 100, 130, 160, 100
t# = Timer - t#, qt# = qTimer - qt#, ot% = oTimer
  - ot%
Color 0
Print AT(1, 1); "Time according to Timer function:
  "; Format(t#, "0.###"); " secs"
Print AT(1, 2); "Time according to oTimer
  function: "; Format(ot% / 1000, "0.###"); " secs"
Print AT(1, 3); "Time according to qTimer
  function: "; Format(qt# * 10, "0.###"); " secs"
Do
  Sleep
Loop Until Me Is Nothing

Proc xdraw(x, y, l, r)
  Local x1#, y1#
  RGBColor RGB(0, 255 - l, 0)
  Draw "ma" x, y, "tt" r, "fd", l
  x1 = Draw(0), y1 = Draw(1)
  If l > .5
    xdraw x1, y1, l * f1, r + w1
    xdraw x1, y1, l * f1, r - w1
    xdraw x1, y1, l * f2, r + w2
  EndIf
EndProc
```

Displays the time taken to draw the graphical image.

Remarks

Timer is not compatible with GFA-BASIC 16, because it now returns seconds, instead of milliseconds. **oTimer** is compatible with GFA-BASIC 16.

See Also

_TimerCount, _TimerFreq

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

_RDTSC Function

Purpose

Returns the number of processor cycles.

Syntax

x = **_RDTSC**

x : int64

Description

With the function **_RDTSC** it is possible to determine how many cycles your program needs to do something by calling the state of the Time Stamp Counter (TSC). In the following for a For-Next

Example

```
OpenW 1
Local Large l1, l2, l3, l4
Local i As Large, x%
For i = 1 To 20
  l1 = _RDTSC : l2 = _RDTSC
  l3 = _RDTSC : l4 = _RDTSC
  Print l2 - l1; l3 - l2; l4 - l3
Next
KeyGet x%
CloseW 1
```

Remarks

Since the introduction of the Pentium the processor provides the cycle counter in an internal 64-bit register. The counter is reset each time the computer is switched on.

Example

```
$StepOff
OpenW 1
Global Double t
Dim l As Large
Global s$
Local i As Register Int
Global sum As Int
Local a$
Print
s$ = Space$(100000)
t = Timer : l = _RDTSC : sum = 0
For i = 1 To Len(s$)
  a$ = Mid$(s$, i, 1)
  sum += Asc(a$)
Next
l = _RDTSC - l : t = Timer - t
Print "Number of cycles: "; l
Print "Time in seconds: "; t
Do :   Sleep : Until Me Is Nothing
```

See Also

Timer, $StepOff, Naked

{Created by Sjouke Hamstra; Last updated: 22/09/2014 by James Gaite}

_ctime Function

Purpose

Converts a **_time** value to a string and adjust for local time zone settings.

Syntax

int = **_ctime** [(V: x%)]

Description

The _**ctime** function converts a time value stored as a _time 32-bit integer into a character string. The timer value is usually obtained from a call to _**time**(), which returns the number of seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time (UTC). The string result produced by ctime contains exactly 26 characters and has the form:

"Wed Jan 02 02:03:55 1980"#10

A 24-hour clock is used. All fields have a constant width. The new line character ('\n' or #10) occupies the last two positions of the string.

The converted character string is also adjusted according to the local time zone settings.

Examples

```
OpenW 1
Local a%, b%, x%
```

```
b% = _time(V:a%) // or: ~_time(V:a%)
Print _ctime(V:a%)
KeyGet x%
CloseW 1
```

And

```
OpenW 1
Local a%, x%
a% = 200000000  // 200 million
Print _ctime(V:a%)
KeyGet x%
CloseW 1
```

Remarks

_ctime and **_time** are implemented for compatibility reasons with C. These functions are restricted to dates between 1970 and 2038.

See Also

_time. Now$

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

ChDir Command

Purpose

sets the current directory.

Syntax

ChDir a$

a$:sexp; name of current directory

Description

ChDir sets the current directory. Since it is impossible to change the drive with **ChDir**, this command always defaults to the current drive. **ChDir** must be followed by the path name. If a$ contains only the backslash ("\"), the change to the root directory of the current drive is performed.

There are two special abbreviations for **ChDir**: "." and "..". "." is an alternative way to define the current subdirectory and ".." for the parent directory. Let's assume that the current subdirectory contains the directory Test, which in turn contains directories A1 and A2. \Test\A1 is the current path. In this case **ChDir** "..\A2" will change the current path to \Test\A2.

Example

```
ChDrive 1      // Drive A is the current drive
ChDir "\Test" // A:\Test
ChDir "A1"    // A:\Test\A1
ChDir "..\A2" // A:\Test\A2
```

See Also

ChDrive, CurDir

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

ChDrive Command

Purpose

Sets the current drive.

Syntax

ChDrive n or n$

n:integer expression
n$:sexp

Description

ChDrive (change drive) sets the current drive. If an input or output command does not contain a drive, all inputs and outputs default to the current drive. n can assume the values from 1 to 16, and these values correspond to drives A to P. Instead of a drive number, **ChDrive** can also take a string whose first character is the drive letter.

Example

```
ChDrive 1      // Drive A is the current drive
ChDir "\Test" // A:\Test
ChDir "A1"    // A:\Test\A1
ChDir "..\A2" // A:\Test\A2
```

Remarks

ChDrive and **_Drive** should be used with much care, because through the increasing use of network drives other notations are used as well. For instance ..//Hallo\.. etc).

See Also

 Drive

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

CurDir Function

Purpose

Returns a **String** representing the current path.

Syntax

CurDir[$]()

Description

Returns the current path for the application. For a network drive the return value won't contain a drive ("\\server\test\test").

Example

```
OpenW 1
Print "Current Directory: "; CurDir()
```

Remarks

Don't use commands or function that require a hard coded drive.

See Also

ChDir, _Drive

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

_Dc2 Function

Purpose

Returns the handle of the Device Context for the **AutoRedraw** window area.

Syntax

h=**_DC2**([w])

h: Handle
w: iexp

Description

_DC2() is available only when **AutoRedraw** = True. **AutoRedraw** uses the second device context to repaint the window. The argument can be a value between 0 and 31 representing a window opened using **OpenW**, **ParentW**, and **ChildW**. Other forms should use the .**hDC2** property of the object.

Example

```
OpenW 1 : AutoRedraw = 1
Print Me.hDC2
Print _DC2(1)
```

Remarks

Implemented for compatibility reasons.

_Dc2(1) is equivalent to **Win_1.hDC2**.

_Dc2() is equivalent to **Me.hDC2**.

See Also

_Dc(), AutoRedraw, hDC, hDC2

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

_Drive Function

Purpose

Specifies the current drive as an integer.

Syntax

% = **_Drive**

%: integer expression

Description

_Drive specifies the current drive as a numeric, e.g. 67 for drive "C". This is the opposite of **ChDrive** d%.
Chr$(**_Drive**) returns the drive as a letter. The following program will determine all available drives:

Example

```
Local i%
For i% = Asc("C") To Asc("Z")
  ChDrive Chr(i%)
  If i% = _Drive
    Print "Drive "; Chr$(_Drive)
  EndIf
Next i%
```

See Also

ChDrive

{Created by Sjouke Hamstra; Last updated: 30/09/2014 by James Gaite}

CopyFile, FileCopy Commands

Purpose

The **CopyFile** function copies an existing file to a new file, with the option of returning an error if the new filename already exists; the **FileCopy** function copies an existing file to a new file without checking the existence of the destination file.

Syntax

CopyFile "source" [**Over** [**To**]] "dest" [, subname[,ident%]]

FileCopy "source" [**To**] "dest" [, subname [, ident%]]

"dest", "source"	*: file names paths.*
subname	*: the name of linked procedure.*
ident%	*: 32-bit Integer*

Description

CopyFile and **FileCopy** copy an existing file "source" to the destination file "dest" in 32KB blocks. By default, **CopyFile** checks first to see if the destination file exists, but this check can be over-ridden by the inclusion of the **Over** or **Over To** clauses; **FileCopy** carries out no such check and will overwrite the destination file if it exists.

To use long filenames (in excess of MAX_PATH length of 260), add "\\?\" before the source and destination filenames.

When specified, *subname* is the name of a **Sub** procedure that is invoked after each copied block (32KB). The **Sub** takes two parameters:

Sub *copyfile*(bytes_copied, ident)

bytes_copied	*: 32-bit Integer.*
ident	*: 32-bit Integer*

The *bytes_copied* argument specifies the number of bytes copied at that moment; for file sizes in excess of _MaxInt, see the third example below. The *ident* variable identifies the **CopyFile**/**FileCopy** command (through the value of *ident%*), allowing the procedure to be used for more than one type of copy operation and, thus, allowing some element of customsation; it is also useful if it is planned to carry out more than one copy operation at any one time. Finally, note that if the program is ended by **End** or **Stop** in the midst of copying a file, the copyfile Sub is not always halted at the same time and may continue working afterwards.

Example

```
Open "c:\test.dat" for Output As # 1 : Close # 1
Try
  CopyFile "c:\test.dat" To "c:\backup.dat"
           // Will cause error if c:\backup.dat
    exists
Catch
  If Exist("c:\backup.dat") Then Kill
    "c:\backup.dat"    // Override safety feature
    (if needed)
  CopyFile "c:\test.dat" To "c:\backup.dat"
```

```
EndCatch
CopyFile "c:\test.dat" Over To "c:\backup.dat"
        // 'Over' prevents an error if
  c:\backup.dat exists
FileCopy "c:\test.dat" To "c:\backup.dat"
       // 'Over' prevents an error if c:\backup.dat
  exists
Kill "c:\test.dat" : Kill "c:\backup.dat"
       // Tidy up line
```

or

```
Dim a(200000) As Int32
BSave "c:\test.dat", V:a(0), 800004
Ocx Label lbl = "Save Progress", 10, 10, 180, 14 :
  lbl.BackColor = RGB(255, 255, 255)
Ocx ProgressBar prg = "", 10, 25, 200, 30
// If c:\backup.dat exists, CopyFile will raise an
  error message
If Exist("c:\backup.dat")
  FileCopy "c:\test.dat" To "c:\backup.dat",
    check_it, 1
Else
  CopyFile "c:\test.dat" To "c:\backup.dat",
    check_it, 2
EndIf
prg.Value = 100
Ocx Command cmd = "Close", 60, 65, 100, 22
Do : Sleep : Until Me Is Nothing
Kill "c:\test.dat" : Kill "c:\backup.dat" // Tidy
  up line

Sub check_it(bytes_copied%, ident%)
  If ident% = 1
    lbl.Caption = "Save Progress using FileCopy:"
  Else If ident% = 2
    lbl.Caption = "Save Progress using CopyFile:"
  EndIf
  prg.Value = 100 * (bytes_copied% / 800004)
  Pause 1 // Included purely to lengthen the time
    the program runs to allow you to see the
    effects of this Sub
EndSub

Sub cmd_Click
  Me.Close
EndSub
```

When the file size is greater than **_MaxInt**, the following workaround can be used:

```
Dim a(200000) As Int32
BSave "c:\test.dat", V:a(0), 800004
// If c:\backup.dat exists, CopyFile will raise an
  error message
If Exist("c:\backup.dat") Then Kill
  "c:\backup.dat"
check_it(0, -1) // Set bytes_count to zero
CopyFile "c:\test.dat" To "c:\backup.dat",
  check_it, 1
check_it(0, -1) // Set bytes_count to zero
FileCopy "c:\test.dat" To "c:\backup.dat",
  check_it, 2
Ocx Command cmd = "Close", 60, 65, 100, 22
Do : Sleep : Until Me Is Nothing
Kill "c:\test.dat" : Kill "c:\backup.dat" // Tidy
  up line

Sub check_it(bytes_copied%, ident%)
  Static bytes_count As Large
  If ident% = -1
    bytes_count = 0
  Else
    bytes_count = bytes_count + (32 * 1024)
    Print AT(1, ident%); "Bytes copied: ";
     bytes_count; "   "
    Pause 1 // Included purely to lengthen the time
     the program runs to allow you to see the
     effects of this Sub
  EndIf
EndSub

Sub cmd_Click
  Me.Close
EndSub
```

Remarks

CopyFile and **FileCopy** can take ':Files' resource file as an argument.

CopyFile is not a GFA-BASIC 32 implementation of the API function *CopyFileEx*(), because it is available only from NT onwards.

See Also

MoveFile

{Created by Sjouke Hamstra; Last updated: 15/01/2016 by James Gaite}

MoveFile Command

Purpose

Moves or renames an existing file.

Syntax

MoveFile source **To** destination

source, destination:sexp

Description

If *source* contains wildcards or *destination* ends with a path separator (****), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination file to create. In either case, three things can happen when an individual file is moved:

If *destination* does not exist, the file gets moved. This is the usual case.

If *destination* is an existing file, an error occurs.

If destination is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any files. The **MoveFile** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.

The *destination* argument can't contain wildcard characters.

Example

```
// Create test file
Open "c:\test.dat" for Output As # 1
Close # 1
// Move to Windows Folder
MoveFile "c:\test.dat" To WinDir$ & "\test.dat"
// Trying to move it a second time will result in
  an error
'
// Tidy up test file
Kill WinDir$ & "\test.dat"
```

Remarks

MoveFile conforms to the MSDOS command Move.

See Also

CopyFile

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

MkDir Command

Purpose

creates a directory.

Syntax

MkDir path$ [**Like** *template$*]

path$:sexp; directory name

Description

MkDir path$ (make directory) creates a directory with name path$.

MkDir path$ **Like** *template*$ creates a new directory with a specified path that retains the attributes of a specified template directory. If the underlying file system supports security on files and directories, the function applies a specified security descriptor to the new directory. The new directory retains the other attributes of the specified template directory.

Example

```
MkDir "C:\TEST"
```

Creates a directory called TEST on drive C

Remarks

The **MkDir...Like** command uses the API function *CreateDirectoryEx*, which allows you to create directories that inherit stream information from other directories. This function is useful, for example, when dealing with Macintosh directories, which have a resource stream that is needed to properly identify directory contents as an attribute.

Some file systems, such as NTFS, support compression or encryption for individual files and directories. On volumes formatted for such a file system, a new directory inherits the compression and encryption attributes of its parent directory.

See Also

RmDir

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

RmDir Command

Purpose

Deletes a directory.

Syntax

RmDir path$

path$:sexp; directory name

Description

RmDir a$ (remove directory) deletes the directory with the name a$, assuming it does not contain any subdirectories.

Example

```
MkDir "c:\TEST"
Print "Directory made: "; Dir("c:\TEST", 16)
RmDir "C:\TEST"  //Deletes the directory TEST on
  drive C.
Print
Print "Directory deleted: "; Dir("c:\TEST", 16)
```

See Also

MkDir

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Name Property

Purpose

Returns the name used in code to identify a form or Ocx control.

Syntax

$ = *object*.**Name**

object:Ocx Object

Description

The default name for new objects is the kind of object plus a unique integer. For example, the first new **Form** object is *frm1*, a new **Command** object is *cmd1*, and the third **TextBox** control you create on a form is *txt3*.

An object's **Name** property must start with a letter and can be a maximum of 40 characters. It can include numbers and underline (_) characters but can't include punctuation or spaces. Forms can't have the same name as another public variable. Although the **Name** property setting can be a keyword, property name, or the name of another object, this can create conflicts in your code.

Note Although GFA-BASIC 32 often uses the **Name** property setting as the default value for the **Caption** and **Text** properties, changing one of these properties doesn't affect the others.

Example

```
Print Me.Name
```

Remarks

The names of from created with **OpenW** and **Dialog** are predefined as **Win_*n*** and **Dlg_*n*** respectively, where n is a number between 0 and 31. The name is introduced in the global variable list and is accessible throughout the program. These variable names can be used in accessing properties, methods, and events. For instance, **Win_1.Name** returns "Win_1". Windows created with a number greater than 31 don't declare global variable names implicitly and can only be accessed using **Form**(n).Name. However, there is no variable name introduced but their name still consists of "Win_n", where n is the window number.

```
OpenW 100
Print Form(100).Name                // Win_100
Do : Sleep : Until Me Is Nothing

Sub Form_Click(index%)
  Print Me.Name, index%            // Win_100   100
EndSub
```

See Also

Form

{Created by Sjouke Hamstra; Last updated: 19/09/2016 by James Gaite}

Name...As and Rename...As Commands

Purpose

Renames a file.

Syntax

Name old$ **As** new$

P>**Rename** old$ **As** new$

old$, new$:sexp; old and new file names

Description

Name...As is synonymous with **Rename...As**, and both rename the specified file.

Example

```
Dim old$ = "C:\TEST.DAT", new$ = "C:\TEST.TXT"
// Create "c:\test.dat"
BSave old$, 100000, 100
Print "Directory showing "; old$
Dir "c:\*.*"
Print
// Rename "c:\test.dat" as "c:\test.txt"
If Exist(new$) Then Kill new$ (* If Test.txt
  exists, Name will cause an error *)
Name old$ As new$
Print "Directory showing "; new$
Dir "c:\*.*"
```

```
Print
// Change "c:\test.txt" back to "c:\test.date"
If Exist(old$) Then Kill old$ (* If Test.txt
 exists, Name will cause an error *)
Rename new$ As old$
Print "Directory showing "; old$
Dir "c:\*.*"
```

Remarks

See Also

Rename As

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

FileDateTime Functions

Purpose

Returns a Date that indicates the date and time when a file was created, accessed or last modified.

Syntax

dt **= FileDateTime** ([Pathname$])

dt **= FileDateTimeAccess** ([Pathname$])

dt **= FileDateTimeCreate** ([Pathname$])

dt:Date

Description

The optional *pathname* argument is a string expression that specifies a file name. The *pathname* may include the directory or folder.

Without an argument the function returns the Date for the last file accessed using **Dir**().

Example

```
OpenW 1
Global d$, p1 As Int32
// Get the path for GfaWin32.exe
Local d$ =
  GetSetting("\\HKEY_CLASSES_ROOT\Applications\GfaW
  in32.exe\shell\open\command", , "")
If Left(d$, 1) = #34 Then d$ = Mid(d$, 2)
```

```
p1 = InStr(d$, #34) : If p1 <> 0 Then d$ =
  Left(d$, p1 - 1)
// Display File Date information
Print "GfaWin32.exe was created: ";
  FileDateTimeCreate(d$)
Print "The last time that GfaWin32.exe was
  modified or moved was: "; FileDateTime(d$)
Print "The last time that GfaWin32.exe was
  accessed was: "; FileDateTimeAccess(d$)
Print
// The same results can be achieved through the
  FileSystemObject
Global Object f, fs
Set fs =
  CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(d$)
Print "GfaWin32.exe was created: "; f.DateCreated
Print "The last time that GfaWin32.exe was
  modified or moved was: "; f.DateLastModified
Print "The last time that GfaWin32.exe was
  accessed was: "; f.DateLastAccessed
Do : Sleep : Until Me Is Nothing
Set f = Nothing : Set fs = Nothing
CloseW 1
```

Remarks

Known Issues

FileDateTimeAccess doesn't always return a time when querying FAT32 files; this bug does not seem to affect **FileDateTimeCreate** or **FileDateTime**.

See Also

FileLen(), SetFileDateTime, SetFileDateTimeAccess, SetFileDateTimeCreate

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

SetFileDateTime, SetFileDateTimeAccess, SetFileDateTimeCreate Command

Purpose

Sets the date and time of the last access of a file.

Syntax

SetFileDateTime file$, date

SetFileDateTimeAccess file$, date

SetFileDateTimeCreate file$, date

file$:sexp
date:Date exp

Description

The **SetFileDateTime** changes the last access time and/or date information assigned to a file. The command doesn't work on write protected files. Internally, it performs an **Open**, which might be blocked by some other application.

The **SetFileDateTimeAccess** changes the access time and/or date information assigned to a file. The command doesn't work on write protected files. Internally, it performs an **Open**, which might be blocked by some other application.

The **SetFileDateTimeCreate** changes the create time and/or date information assigned to a file. The command doesn't work on write protected files. Internally, it performs an **Open**, which might be blocked by some other application.

Example

```
// Create Test file
BSave App.Path & "\Test.Dat", 100000, 100
Debug.Show
// Set file times
SetFileDateTime App.Path & "\Test.Dat",
  #20.12.2006#
SetFileDateTimeCreate App.Path & "\Test.Dat",
  #20.12.2001#
SetFileDateTimeAccess App.Path & "\Test.Dat",
  #20.12.2003#
// Show file times
Trace FileDateTime(App.Path & "\Test.Dat")
Trace FileDateTimeCreate(App.Path & "\Test.Dat")
Trace FileDateTimeAccess(App.Path & "\Test.Dat")
// Tidy Up
Kill App.Path & "\Test.Dat"
```

Remarks

Windows 95 ignores the time part.

See Also

Touch

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Touch Command

Purpose

Updates the time and date stamps of a file with current values.

Syntax

Touch #n

n:iexp, channel

Description

Touch[#]n works only on files already opened with **Open** by making their time and date stamps current. The time and date stamps of the open file are set to values obtained from the system clock.

Example

```
Local f$ = App.Path + "\Test.temp"
Local a%, i%
OpenW # 1
Open f$ for Output As # 1
For i% = 1 To 20
  Print # 1, Format(i%)
Next i%
Close # 1
Files f$
//
For i% = 1 To 9  '9 Second Pause
```

```
  Print AT(1, 3); "A short pause -"; 10 - i%; "
    seconds to go"
  Delay 1
Next i%
Print AT(1, 3); "Pause over" & Space(100) : Print
//
Open f$ for Update As # 1
Touch # 1
Close # 1
Files f$
Kill f$
```

Opens the Test.temp file and writes the numbers from 1 to 20 to it. The **Files** is then used to print, among others, the time and date stamps.

A 10 second pause follows next. The time and date stamp of the Test.temp file are then updated with **Touch** and printed again using **Files**.

See Also

SetAttr, SetFileDateTime, SetFileDateTimeAcces, SetFileDateTimeCreate

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

GetAttr, SetAttr Functions

Syntax

% = **GetAttr(***pathname*)

success% = **SetAttr**(*pathname*, attr) (function)

SetAttr *pathname*, attr (command)

Included for compatibilty with GFA-BASIC 16:

% = **FGATTR** (*pathname*) (same as **GetAttr**)

success& = **FSATTR** (*pathname*, attr&) (same as **SetAttr** function)

Description

The function **GetAttr** returns the attributes of a file or a directory while **SetAttr** sets them. Following constants (values) are predefined:

FILE_ATTRIBUTE_NORMAL (0) - Normal file

FILE_ATTRIBUTE_READONLY(1) - Read-Only (write protected)

FILE_ATTRIBUTE_HIDDEN (2) - Hidden

FILE_ATTRIBUTE_SYSTEM (4) - System

FILE_ATTRIBUTE_DIRECTORY (16) - Directory

FILE_ATTRIBUTE_ARCHIVE (32) - Archive (reserved for Backups).

FILE_ATTRIBUTE_TEMPORARY (256) - Temporary file

FILE_ATTRIBUTE_OFFLINE (4096) - Offline

More values may be returned. These values can not be set using **SetAttr**, though.

64	encrypted file, set by *EncryptFile*
512	Joke file (file with holes)
1024	Reparse
2048	compressed
8192	Not contended index

If either of the functions fail, the return value is -1; the command version of **SetAttr** should be used within a **Try-Catch** block to catch any possible errors.

With **GetAttr**, to determine which attributes are set, use the **And** operator to perform a bitwise comparison of the value returned by the **GetAttr** function and the value of the individual file attribute you want. If the result is not zero, that attribute is set for the named file. For example, the return value of the following **And** expression is 16 if the directory exists:

```
If GetAttr("directory") And 16 Then // Directory
 exist!
```

GetAttr() returns the attributes of the last **Dir**[$].

Example

```
OpenW 1
```

```
// Read the contents of the current path
// and show: attribute,
// size of a file in KB, date, time, name
FullW 1
PrintScroll = True       ' activate scrolling
Local file$, a$, b$
Local Attr As Integer
file$ = Dir$("*", &H16)
While Len(file$) : a$ = ""
  Attr = GetAttr(file$)
  a$ = a$ + Iif(Attr And 32, "A", "-")
  a$ = a$ + Iif(Attr And 16, "D", "-")
  a$ = a$ + Iif(Attr And 4, "S", "-")
  a$ = a$ + Iif(Attr And 2, "H", "-")
  a$ = a$ + Iif(Attr And 1, "R", "-")
  If Attr And 16 Then
    a$ = a$ + " <Dir>"
  Else
    b$ = Str$(FileLen(file$))
    b$ = Space$(8 - Len(b$)) + b$
    a$ = a$ + Format(FileLen(file$), "* #######0")
    a$ = a$ + b$
  End If
  a$ = a$ + "  "
  If file$ <> "." And file$ <> ".." Then
    a$ = a$ + Format(FileDateTime(file$),
     "dd.mm.yyyy hh:nn:ss ")
  End If
  b$ = ShortFileName()
  If b$ = "" : b$ = file$ : EndIf
  a$ = a$ + Str$(b$, 14) + "  "
  Print a$
  file$ = Dir
Wend
Do : Sleep : Until Me Is Nothing
```

Set the write protecting of the file "Test1.Dat"

```
Local a% = 25
Print App.Path & "\Test1.Dat"
BSave App.Path & "\Test1.Dat", V:a%, 4
SetAttr App.Path & "\Test1.Dat", GetAttr(App.Path
  & "\Test1.Dat") | 1
If GetAttr(App.Path & "\Test1.Dat") And 1
  Print "write protected"
Else
  Print "not write protected!"
EndIf
SetAttr App.Path & "\Test1.Dat", GetAttr(App.Path
  & "\Test1.Dat") Xor 1
If GetAttr(App.Path & "\Test1.Dat") And 1
  Print "write protected"
Else
  Print " not write protected!"
EndIf
Kill App.Path & "\Test1.Dat"  // Tidy up line
```

Remarks

To remove and set a write protection of a backup:

```
SetAttr "important.Bak", 0
CopyFile "important.Dat" Over To "important.Bak"
SetAttr "important.Bak", 1      ' activate write
  protection
```

The **GetAttr** function corresponds to the *GetFileAttributes* API function.

The **SetAttr** command corresponds to the *SetFileAttributes* API function

See Also

Dir, FileAttr, SetFileDateTime, SetFileDateTimeAcces, SetFileDateTimeCreate, Touch.

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

Exist Function

Purpose

Determines if a particular file exists.

Syntax

Exist(a$)

a$:sexp; path name of a file

Description

The **Exist**(a$) function determines if a particular file exists in the path name specified in a$. **Exist**() returns -1 (True) if this file exists or 0 (False) if not.

Example

```
OpenW 1
Global a$, c$, a%, d%, x%
a$ = "C:\TEST.DAT"
If Exist(a$)
  Open a$ for Input As # 1
  Do Until EOF(# 1)
    Input # 1, c$
    Print c$
  Loop
  Close # 1
Else
  Alert 1, "File not found", 1, "ok", d%
EndIf
```

Determines if the file TEST.DAT exists on drive C and, if it does, reads the file in.

Remarks

To test for a directory use **GetAttr**("dir") **And** 16 == 16

See Also

GetAttr

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

FileLen Function

Purpose

Returns a Long or a Large specifying the length of a file.

Syntax

sizeLarge = **FileLen** ([Pathname$]) *

sizeInt = **FileLen%** ([Pathname$])

sizeLarge:int64
sizeInt:int32

* actually returns a 32-bit Integer - see Known Issues below

Description

The optional *pathname* argument is a string expression that specifies a file name. The *pathname* may include the directory or folder.

Without an argument the function returns the Date for the last file accessed using **Dir**().

Example

```
OpenW 1
Global d$, p1 As Int32
// Get the path for GfaWin32.exe
Local d$ =
  GetSetting("\\HKEY_CLASSES_ROOT\Applications\GfaW
  in32.exe\shell\open\command", , "")
```

```
If Left(d$, 1) = #34 Then d$ = Mid(d$, 2)
p1 = InStr(d$, #34) : If p1 <> 0 Then d$ =
  Left(d$, p1 - 1)
// Display File Date information
Print "The length of GfaWin32.exe in bytes is: ";
  FileLen(d$)
Print
// The same results can be achieved through the
  FileSystemObject
Global Object f, fs
Set fs =
  CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(d$)
Print "The length of GfaWin32.exe in bytes is: ";
  f.size
Do : Sleep : Until Me Is Nothing
Set f = Nothing : Set fs = Nothing
CloseW 1
```

Remarks

FileLen% is GFA-BASIC 16 compatible, because it returns a 32-bit integer. However, it will return the wrong result for files larger than 2 GB.

Known Issues

It has been reported that, on some computers, FileLen returns a 32-bit Integer, not a Large 64-bit integer; it has also been reported that this behaviour can be intermittent, even on the same computer. What appears to be happening is GB32 uses FindFirstFile() to retrieve the file length, namely through the FileSizeHi and FileSizeLo DWord (or Long) properties of the Win32_Find_Data object: when FileSizeLo is returned as a positive value, then GB32 returns the correct file length; however, when FileSizeLo is negative, GB32 ignores the FileSizeHi and just returns the negative FileSizeLo value.

If you encounter this problem, use the Windows FileSystemObject as shown below as a workaround to get the file length of large files:

```
Dim myFSO As Object, myFile As Object
Set myFSO =
 CreateObject("Scripting.FileSystemObject")
Set myFile = myFSO.getfile("[Full_File_Path]")
Print myFile.size
```

See Also

FileDateTime(), FileDateTimeAccess(), FileDateTimeCreate(), SetFileDateTime, SetFileDateTimeAccess, SetFileDateTimeCreate

{Created by Sjouke Hamstra; Last updated: 15/12/2014 by James Gaite}

Dir Function

Purpose

Returns a **String** representing the name of a file, directory or folder that matches a specified file attribute(s), or the volume label of a drive.

Syntax

**Dir[$]**[(fname$ [,attr])

fname : svar
attr : ivar

Description

The *fname* specifies a file name - this may include a directory (folder) and drive letter. A zero-length string ("") is returned if *fname* is not found. **Dir** supports the use of multiple character (*****) and single character (**?**) wildcards to specify multiple files.

The *attr* parameter specifies the file attribute(s) of the files to include. If omitted, **Dir** returns files that match *pathname* but have no attributes. Normally, if *attr* is not used or attr is 0 or 1, only all non-hidden files and read only one are shown.

Following constants (values) are predefined:

FILE_ATTRIBUTE_NORMAL (0)	normal file
FILE_ATTRIBUTE_READONLY(1)	Read-Only (write protected)

FILE_ATTRIBUTE_HIDDEN (2)	Hidden
FILE_ATTRIBUTE_SYSTEM (4)	System
FILE_ATTRIBUTE_DIRECTORY (16)	Directory
FILE_ATTRIBUTE_ARCHIVE (32)	Archive (reserved for Backups).
FILE_ATTRIBUTE_TEMPORARY (256)	temporary file
FILE_ATTRIBUTE_OFFLINE (4096)	offline

If you require the hidden and/or the system files, you have to add 2, 4 or 6 to the *attr* value. For example; **Dir$**("*", 6) shows the hidden files also.

Dir without parameters gets the next file. When the last file is reached **Dir** returns an empty string.

Example

To display the first file in a directory:

```
OpenW 1
Local a%
PrintScroll = True
Print Dir("c:\Windows\*.dll")
```

Display all files in a directory (comparable to the MSDOS dir /a/b command):

```
OpenW 1
Local contents$, a%
PrintScroll = True
contents$ = Dir("c:\windows\*", $16)
While Len(contents$)
  Print contents$
  contents$ = Dir$
Wend
```

Remarks

It is possible to combine the attributes with a binary **Or**. In this way **Dir**$("*", 16 | 6) lists all normal and hidden files, and names of (hidden) directories, including "." and ".."".

// Directory - example

```
Global file$, a$, b$, Attr As Int
file$ = Dir$("*", $16)
While Len(file$) : a$ = "" : Attr = GetAttr()
  a$ = a$ + (Attr And 32 ? "A" : "-")
  a$ = a$ + (Attr And 16 ? "D" : "-")
  a$ = a$ + (Attr And 4 ? "S" : "-")
  a$ = a$ + (Attr And 2 ? "H" : "-")
  a$ = a$ + (Attr And 1 ? "R" : "-")
  If Attr And 16 Then
    a$ = a$ + " <Dir>"
  Else
    a$ = a$ + Str$(FileLen(), 8)
  EndIf
  a$ = a$ + " "
  a$ = a$ + Date$(FileDateTime()) + _
    " " + Time$(FileDateTime()) + " "
  // extension: time of the last access(date)
  //a$ = a$ + Date$(FileDateTimeAccess() + " "
  b$ = ShortFileName()
  If b$ = "" : b$ = file$ : EndIf
  a$ = a$ + Str$(b$, 14) + " "
  // Str$(string, cnt) returns a string which will
    filled with
  // spaces, same like: Right$(string, cnt, 32)
  // a$=a$+str$(ShortFileName(), 14) + " "
  // ShortFileName() returns an empty string if
    file$ will
  // fit to the MS-DOS name
  a$ = a$ + file$
  Print a$
  file$ = Dir
Wend
```

This program creates the same output as the MS-DOS command DIR /a, similar to that shown below:

A---- 1000 30.10.1995 00:00:00 TEST.DAT Test.Dat

Description:

- A the archive bit is set (identification for PKZIP, ARJ, RAR, BACKUP etc.)
- - no directory (not D)
- - no hidden file (not H)
- - no system file (not S)
- - not Read-Only (write protected) (not R)
- 1000 the length of the file is 1000 bytes
- 30.10.1995 date of the file
- 00:00:00 mid night
- TEST.DAT name of he file - MS-DOS convention (8.3)
- Test.Dat name of the file - Windows 32 bit file name (small/large, long......)

```
// The same program now for VB,
// it works both in GFA-BASIC 32 and VB
Global file$, a$, b$, Attr As Int
file$ = Dir$("*", &H16)
While Len(file$)
  a$ = "" : Attr = GetAttr(file$)
  a$ = a$ + Iif(Attr And 32, "A", "-")
  a$ = a$ + Iif(Attr And 16, "D", "-")
  a$ = a$ + Iif(Attr And 4, "S", "-")
  a$ = a$ + Iif(Attr And 2, "H", "-")
  a$ = a$ + Iif(Attr And 1, "R", "-")
  If Attr And 16 Then
    a$ = a$ + " <Dir>"
  Else
    b$ = Str$(FileLen(file$))
    b$ = Space$(8 - Len(b$)) + b$
    'a$ = a$ + Format(FileLen(file$), _
      ' "* #######0")
    a$ = a$ + b$
  EndIf
  a$ = a$ + " "
  If file$ <> "." And file$ <> ".." Then _
    a$ = a$ + Format(FileDateTime(file$), _
    "dd.mm.yyyy hh:nn:ss ")
  'b$ = ShortFileName()
  'If b$ = "" : b$ = file$ : EndIf
  'a$ = a$ + Str$(b$, 14) + " "
  a$ = a$ + file$
  Print a$
  file$ = Dir
Wend
```

To list the contents of the subdirectories as well use **DirPush** and **DirPop**.

See Also

DirPush, DirPop, DirPopAll, LongFileName() , ShortFileName(), FileDateTime$(), GetAttr(), FileLen(),ChDir, CurDir(), Dir To

{Created by Sjouke Hamstra; Last updated: 30/09/2014 by James Gaite}

DirPush, DirPop and DirPopAll Commands

Purpose

Moves the current Dir() settings onto and from the stack.

Syntax

DirPop DirPopAll
DirPush

Description

When moving from a folder into a sub-folder using the **Dir** command, rather than having to recreate the original settings (and then navigate to the current folder again) when returning to the parent folder, it is possible to use **DirPush** to store them on the stack, and **DirPop** to retrieve them when required. Once you have finished, you can use **DirPopAll** to remove any remaining settings and clear the stack.

Example

See the sample program "RecurseDir2.g32" in GFABASIC32\gb32\Samples\Files

Remark

A stack is a LIFO system, where the last element stored is retrieved first (last-in-first-out). So, if you have saved the path 20 times with **DirPush** you can go back to the eleventh instance by invoking **DirPop** 10 times.

See Also

Dir()

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Long/ShortFileName, Long/ShortPathName Functions

Action

Return a long filename and long path name of a file.

Syntax

LongFileName[$]([file$])

LongPathName[$]([file$])

ShortFileName[$]([file$])

ShortPathName[$]([file$])

file$:filename

Description

With the function **LongFileName**() you can determine a long filename or directory from a short name, for example: "StartMenu" instead of its short name "STARTM~1". **ShortFileName** does the reverse (see Known Issues).

LongFileName() also returns the pathname of the last call of **Dir**[$].

With the function **LongPathName**() you determine the long path name of a file, for example: "c:\example-folder\test‟ instead of "c:\exam~1\test" (see Known Issues); **ShortPathName** does the reverse.

Example

```
// Create two test files
// Files must exist or the functions return a File
  Name error
Local f1$ = App.scSpecialDir(39) & "\tinyname.bmp"
Local f2$ = App.scSpecialDir(39) &
  "\reallylongfilename.bmp"
BSave f1$, 100000, 100 : BSave f2$, 100000, 100
Local f3$ = ShortPathName(f2$)
// Show the results in the Debug screen
Debug.Show
Trace f1$
Trace LongFileName(f1$)
Trace ShortFileName(f1$) // Error: Returns a blank
  - see known issues below
Trace LongPathName(f1$)
Trace ShortPathName(f1$)
Debug.Print
Trace f2$
Trace ShortFileName(f2$) // Acts correctly - see
  known issues below
Trace ShortPathName(f2$)
Debug.Print
Trace f3$
Trace LongFileName(f3$)
Trace LongPathName(f3$) // Error: Returns the
  Short Path - see known issues below
// Remove test files
Kill f1$
Kill f2$
```

Known Issues

If the filename in *file$* fits within the old 8.3 format (filename <=8; extension <=3) then **ShortFileName** returns a blank rather than the filename. There are two possible workarounds for this:

1. Create a function such as the one below which reverts to the original filename if **ShortFileName** returns a blank.

```
Local f1$ = App.scSpecialDir(39) &
  "\tinyname.bmp"
Local f2$ = App.scSpecialDir(39) &
  "\reallylongfilename.bmp"
BSave f1$, 100000, 100 : BSave f2$, 100000, 100
Print GetShortName(f1$)
Print GetShortName(f2$)
Kill f1$ : Kill f2$

Function GetShortName(file$)
  If ShortFileName(file$) = "" Then Return
    Upper(file$)
  Return ShortFileName(file$)
EndFunc
```

2. Use the result from ShortPathName() as in the example below:

```
Local f1$ = App.scSpecialDir(39) &
  "\tinyname.bmp"
Local f2$ = App.scSpecialDir(39) &
  "\reallylongfilename.bmp"
BSave f1$, 100000, 100 : BSave f2$, 100000, 100
Print GetShortName(f1$)
Print GetShortName(f2$)
Kill f1$ : Kill f2$

Function GetShortName(file$)
  Local slen As Byte, sf$
```

```
    sf$ = ShortPathName(file$)
    slen = RInStr(sf$, "\")
    Return Upper(Mid(sf$, slen + 1))
  EndFunc
```

LongPathName does not return the long path name as stated; an example of this and a workaround using the GetLongPathName() API is below:

```
Local f2$ = App.scSpecialDir(39) &
  "\reallylongfilename.bmp"
BSave f2$, 100000, 100
Local f3$ = ShortPathName(f2$), f4$ = Space(255)
Print LongPathName(f3$) // Error
Print GetLongPath(f3$)
Kill f2$

Function GetLongPath(file$)
  Declare Function GetLongPathName Lib "kernel32"
    Alias "GetLongPathNameA" (ByVal lpszShortPath
    As String, ByVal lpszLongPath As String, ByVal
    cchBuffer As Long) As Long
  '
  Local fp$ = Space(255), flen =
    GetLongPathName(file$, fp$, 255)
  Return Left(fp$, flen)
EndFunc
```

See Also

Dir, ShortProgName(), App

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

ShortProgName and ProgName Function

Purpose

Returns the 'short' name of the current program.

Syntax

$ = **ShortProgName**[$]()

$ = **ProgName**[$]

Description

ProgName[$] returns the directory of the current running application. In the IDE the name of GFA-BASIC 32 is returned.

ShortProgName returns the 'short' MSDOS name (8.3 characters) name of the program.

Example

```
Debug.Show
Trace ProgName()
Trace ShortProgName()
```

Remarks

See Also

LongFileName(), LongPathName(), ShortFileName(), ShortPathName(), ShortProgName(), App

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

WinDir Function

Purpose

Returns the Windows directory.

Syntax

$ = **WinDir**[$]

Description

Returns the Windows directory without an ending backslash.

Example

```
Message WinDir
```

Remarks

The system directories have different names on different machines and OSs. For often used directories GFA-BASIC 32 provides **WinDir**, **SysDir**, and **TempDir** to return the specific directories. Other Shell related directories can be obtained using the App object properties like **scPrograms** and **scSpecialDir**.

See Also

SysDir, TempDir, scSpecialDir

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

SysDir Function

Purpose

Returns the Windows system directory.

Syntax

$ = **SysDir**[$]

Description

Returns the Windows directory without an ending backslash.

Example

```
Message SysDir
```

Remarks

The system directories have different names on different machines and OSs. For often used directories GFA-BASIC 32 provides **WinDir**, **SysDir**, and **TempDir** to return the specific directories.

See Also

WinDir, TempDir, scSpecialDir

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

TempDir Function

Purpose

Returns the path of the directory designated for temporary files.

Syntax

$ = **TempDir**[$]

Description

Returns a string specifying the temporary file path. The returned string ends with a backslash, for example, C:\TEMP\.

The **TempDir** function checks for the existence of environment variables in the following order and uses the first path found:

The path specified by the TMP environment variable (%TMP%).

The path specified by the TEMP environment variable (%TEMP%).

The path specified by the USERPROFILE environment variable (%USERPROFILE%).

The Windows directory.

Note that the function does not verify that the path exists.

Windows Me/98/95: If TMP and TEMP are not set to a valid path, **TempDir** uses the current directory.

Example

```
Message TempDir
```

Remarks

The system directories have different names on different machines and OSs. For often used directories GFA-BASIC 32 provides **WinDir**, **SysDir**, and **TempDir** to return the specific directories.

See Also

SysDir, WinDir, TempFileName, scSpecialDir

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

TempFileName Function

Purpose

Creates a name for a temporary file.

Syntax

file$ = TempFileName(prefix$ [, extension$])

Description

TempFileName tries to create a temporary file in the user's %TEMP% directory and returns the name in file$ - if file$ is empty (""), then the operation failed.

The **TempFileName** function is a shortcut for the *GetTempFileName* API function which will only create the temporary file if it has a unique filename. Through the API, GFA Basic creates a temporary filename which is a concatenation of a prefix string (if prefix$ <> ""), a hexadecimal string derived from the current system time, and a specified extension (or .tmp if none is supplied in extension$).

The *prefix*$ argument may be left empty ("") so that the filename part is entirely made up of a unique hexadecimal value.

Example

```
Trace TempFileName("")
Trace TempFileName("~", "dat")
Trace TempFileName("gfa")
```

```
Global Handle hCur =
  InlLoadCursor("C:\Windows\Cursors\aero_busy.ani")

Function InlLoadCursor(fname$) As Handle
  Dim path$ = TempFileName("gfa")
  Trace fname$ : Trace path$
  CopyFile fname$ Over To path$
  InlLoadCursor = LoadCursorFromFile(path$)
  KillTempFile path$
EndFunc
```

Remarks

File systems attempt to keep all of the data in memory for quicker access rather than flushing the data back to mass storage. A temporary file should be deleted by the application as soon as it is no longer needed.

A file created with the **TempFileName** function is automatically deleted when the programs exits. **KillTempFile** is used when a temporary file is to be deleted explicitly.

See Also

KillTempFile, LoadBmp

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

KillTempFile Command

Purpose

Deletes a temporary file generated with **TempFileName**()

Syntax

KillTempFile path$

path$:sexp; path name

Description

KillTempFile path$ deletes the file whose pathname is given in path$.

Example

```
Local path$ = TempFileName("")
Print path$
KillTempFile path$
Print Exist(path$)
```

Remarks

A file created with the **TempFileName** function is automatically deleted when the programs exits. **KillTempFile** is used when a temporary file is to be deleted explicitly.

See Also

TempFileName

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

Raise, Throw, Clear Methods

Purpose

Methods and property to cause a runtime error to be thrown.

Syntax

*Err***.Raise** *Number[, Source[, Description[, HelpFile[, HelpContext]]]]*

*Err***.Throw**

*Err***.Clear**

Description

The **Raise** method allows you to generate an user-defined error in your code.

number - A Long integer that identifies the nature of the error. GFA-BASIC 32 errors are in the range 0-141.

source - A string expression naming the object or application that originally generated the error.

description - A string expression describing the error. If unspecified, the value in number is examined. If it can be mapped to a GFA-BASIC 32 run-time error code, a string provided by GFA-BASIC 32 is used as description. If there is no GFA-BASIC 32 error corresponding to number, a generic error message is used.

helpfile - The fully qualified path to the Help file in which help on this error can be found.

helpcontext - The context ID identifying a topic within helpfile that provides help for the error.

Note that only the first parameter, *Number*, is required. If you use **Raise**, however, without specifying some arguments, and the property settings of the **Err** object contain values that have not been cleared, those values become the values for your error.

When setting the **Number** property to your own error code, you may add your error code number to the constant **basObjectError** ($800A0000) to simulate a COM error. For example, to generate the error number 10, assign basObjectError + 10 to the **Number** property.

Use **Clear** to explicitly clear the **Err** object after an error has been handled, for example, when you use deferred error handling with **On Error Resume**. The **Clear** method is called automatically whenever any of the following statements is executed: **Try**, **Resume**, **Exit Sub**, **Exit Function**, **On Error** statement

The **Throw** method throws the error back to the next **Try**/**Catch** block. This method allows you to throw a locally created exception in a subroutine. If you try to throw an error that you have just caught, it will normally go out of scope and be deleted. With **Throw**, the error is passed correctly to the calling subroutine.

Note - **Throw** doesn't work as documented. It does generate an error, but the content of **Err** is cleared (which isn't strange in the context of the implicitly invoked **Clear** method on subroutine exit!).

Example

```
OpenW # 1
Try
  RaiseMe
Catch
  MsgBox Err &  " - " & Err.Description, MB_OK,
    "Error in " & Err.Source
EndCatch
CloseW 1

Procedure RaiseMe
  Dim a$ = "1"
  Prompt "Raise an Error", "Which error should be
    shown?", a$
  Try
    Err.Raise Val(a$), "RaiseMe"
  Catch
    MsgBox Err & " - " & Err.Description & #10 _
      "Throw again.", , "Error in " & Err.Source
    Err.Throw
  EndCatch
EndProc
```

Remarks

The **Source** property returns or sets a string specifying the name of the object or application that originally generated the error. For GFA-BASIC 32 runtime errors it is "GFA-BASIC 32", for OLE Automation errors it is the COM program name. When generating an error from code, **Source** is your application’s program name.

See Err$ for a list of errors and exception codes.

See **HResult** for a list of COM error codes.

Err.**Raise** *number* is identical to **Error** *number*.

See Also

Err Object, Source, Error, Err$, HResult

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Error Command

Purpose

Triggers an error.

Syntax

Error n

n:integer expression

Description

Error n raises an error with the number n (see the list of error messages in Err$).

Example

```
Dim a%
OpenW # 1
Input "Which error should be shown";a%
Try
  Error a%
Catch
  Print "This was the error "; Err, Error$
EndCatch
```

Remarks

Error n is a short form for **Err.Raise** n. The usage of the **Err** object will offer the equivalent way to handle errors under GFA-BASIC 32.

See Also

Err Object, Error$, Err$, SysErr

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

Err$, Error$ Functions

Purpose

Returns the error number and text.

Syntax

Err$ [(i)]

Error$ [(i)]

i: integer expression

Description

The **Err$**(i) function returns the string containing the GFA-BASIC error message for code number i. Without an argument **Err**$ returns the string for the last error.

The first 152 error numbers are reserved by GFA-BASIC 32. Hardware exceptions are mostly translated to GFA-BASIC 32 errors. When a GFA-BASIC 32 error results from an exception, the exception number constant and the meaning is mentioned as well.

Error$ is a synonym of **Err**$ and the two are interchangeable.

Err	**Err$**
1	Divide by zero
2	Overflow
3	Parameter invalid
4	Error at Power

5	Error at Sin
6	Error at Cos
7	Error at Tan
8	Error at Fact
9	Error at Combin
10	Error at Variant
11	Error at Bessel function
12	Out of memory
13	Out of string memory
14	String len too big or negative
15	File name
16	File number
17	File not open
18	File number in use
19	File read error
20	File write error
21	File write error (partial written)
22	End Of File reached
23	Open...for Random...Len= mismatch
24	SEEK: no seek allowed
25	LOCK: can't lock
26	UNLOCK: can't unlock (param mismatch?)
27	Parameter SPC: 0 < x < 1000
28	Parameter TAB: 0 < x < 256
29	Declare: library not found
30	Declare: dll not found
31	Error at Kill(File)
32	Error at (Re)Name/ MoveFile
33	Error at CopyFile, FileCopy
34	Error at ChDir

35	Error at MkDir
36	Error at RmDir
37	Error at DFree
38	Array already DIMed
39	Array Index (DIM) too big
40	Arraysize (DIM) too big
41	Parameter at (Q)ROUNDC
42	Bad Format
43	Bad data for Unpack
44	Problem with Joystick-window
45	Error with variant
46	Error with object *Check **HResult** for detailed information on the error.*
47	Variant is not an Object
48	Object is not a Control
49	Object is not a Font
50	Object is not a Picture
51	Object is not a Form
52	Variant type?
53	Stackpointer at PasCall *Might be the result of a wrong ret instruction in assembler code. A call through a function pointer is guarded with a structured exception handling mechanism, so that an error in the called function is trapped. GFA-BASIC 32 then generates error 53. This error can also come up when the function is called using **StdCall** and others.*
54	Address for mFree()
55	Address for mShrink()
56	Error at DatePart
57	Parameter missing

58	Recursion
59	QBDraw?
60	Internal Error
61	Unknown char in Unicode string *GFA-BASIC 32 uses its own (faster) Unicode char conversion functions. An error with conversion results in the error. The conversion functions are heavily used throughout the runtime.*
62	Index out of range (array in variant)
63	Array() in Variant not one dimensional
64	No Array() in Variant
65	VT_UNKNOWN not supported now
66	The object is Nothing
67	Field needs Random File
68	Field bad size (0)
69	Field: bad size (too big)
70	Field total size not matches random len
71	Put #/Get # without Field and without variable
72	Field string len changed
73	The Hash[] is empty
74	Hash[% i starts at 1]
75	Hash[% index too big]
76	Hash["key not found"]
77	Hash[] Internal Error 1 (Version?)
78	Hash[] Internal Error 2 (Memory?)
79	Hash["key already exists"]
80	Hash["empty key not allowed"]
81	Null not allowed
82	(R)InStr startpos must be a simple number
83	(R)InStr 1st and 3rd parameter are simple numbers
84	Parameter mismatch for Mat op

85	Matrix size mismatch
86	Matrix type mismatch (Single and Double)
87	The matrix is not square
88	The inverse matrix could not be determined
89	Type mismatch
90	Not Implemented (now?), probably to be done
91	Read: out of data
92	Read: no data
93	Guard-Page-Violation (Stack Error)
94	Datatype-Misalignment *EXCEPTION_DATATYPE_MISALIGNMENT: The thread tried to read or write data that is misaligned on hardware that does not provide alignment. For example, 16-bit values must be aligned on 2-byte boundaries; 32-bit values on 4-byte boundaries, and so on.*
95	Breakpoint (Int 3 = Monitor) *EXCEPTION_BREAKPOINT: A breakpoint was encountered.*
96	Single-Step (Debugger) *EXCEPTION_SINGLE_STEP: A trace trap or other single-instruction mechanism signaled that one instruction has been executed.*
97	Access-Violation *EXCEPTION_ACCESS_VIOLATION: The thread tried to read from or write to a virtual address for which it does not have the appropriate access.*
98	In-Page-Error *EXCEPTION_IN_PAGE_ERROR: The thread tried to access a page that was not present, and the system was unable to load the page. For example, this exception might occur if a network connection is lost while running a program over the network.*

99	No-Memory
100	Invalid Assembler Instruction (Illegal-Instruction) *EXCEPTION_ILLEGAL_INSTRUCTION: The thread tried to execute an invalid instruction.*
101	Noncontinuable-Exception *EXCEPTION_NONCONTINUABLE_EXCEPTION : The thread tried to continue execution after a non-continuable exception occurred.*
102	Invalid-Disposition *EXCEPTION_INVALID_DISPOSITION: An exception handler returned an invalid disposition to the exception dispatcher. Programmers using a high-level language such as C (and GFA-BASIC 32) should never encounter this exception.*
103	Array-Bounds-Exceeded *EXCEPTION_ARRAY_BOUNDS_EXCEEDED: The thread tried to access an array element that is out of bounds and the underlying hardware supports bounds checking.*
104	Float-Denormal-Operand *EXCEPTION_FLT_DENORMAL_OPERAND: One of the operands in a floating-point operation is denormal. A denormal value is one that is too small to represent as a standard floating-point value.*
105	Float-Divide-By-Zero *EXCEPTION_FLT_DIVIDE_BY_ZERO: The thread tried to divide a floating-point value by a floating-point divisor of zero.*
106	Float-Inexact-Result *EXCEPTION_FLT_INEXACT_RESULT: The result of a floating-point operation cannot be represented exactly as a decimal fraction.*
107	Float-Invalid-Operation *EXCEPTION_FLT_INVALID_OPERATION: This*

exception represents any floating-point exception not included in this list.

108 Float-Overflow
EXCEPTION_FLT_OVERFLOW: The exponent of a floating-point operation is greater than the magnitude allowed by the corresponding type.

109 Float-Stack-Check
EXCEPTION_FLT_STACK_CHECK: The stack overflowed or underflowed as the result of a floating-point operation.

110 Float-Underflow
EXCEPTION_FLT_UNDERFLOW: The exponent of a floating-point operation is less than the magnitude allowed by the corresponding type.

111 Integer-Divide-By-Zero
EXCEPTION_INT_DIVIDE_BY_ZERO: The thread tried to divide an integer value by an integer divisor of zero.

112 Integer-Overflow
EXCEPTION_INT_OVERFLOW: The result of an integer operation caused a carry out of the most significant bit of the result.

113 Privileged-Instruction (I/O Ports for NT)
EXCEPTION_PRIV_INSTRUCTION: The thread tried to execute an instruction whose operation is not allowed in the current machine mode.

114 Stack-Overflow
EXCEPTION_STACK_OVERFLOW: The thread used up its stack.

115 Control-C-Exit
DBG_CONTROL_C: ctrl+c is input to a console process that handles ctrl+c signals and is being debugged. This exception code is not meant to be handled by applications. It is raised only for the

	benefit of the debugger, and is raised only when a debugger is attached to the console process.
116	For Each: this object is not a collection
117	Object type mismatch
118	Wrong type of object for Dim .. As New Type
119	Error on FreeBmp
120	MiMeTo format error
121	No Tool help functions, Windows 95/98/NT 5.0 required
122	Index out of range (ParamArray)
123	Cannot create OCX/Form
124	Owner change not allowed
125	No shell32.dll found
126	Insert/Delete: array not one dimension
127	Insert/Delete: array bound exceeded
128	Insert/Delete: not for boolean array
129	MCI error message
130	uudecode format error
131	Array type error (matrix only double/single)
132	Array dim error (matrix - only 1 and 2 dim)
133	FileOp not for CON:
134	FileOp not for LPT:
135	PolyLine/PolyFill not for Variant/Boolean Arrays
136	The Ocx array is empty
137	Ocx(Index bad)
138	MdiChildWindow needs MdiParentWindow
139	Error on System
140	reStop
141	This API function exists in 16 Bit only
142	Error when writing to the registry
143	Error creating registry key

144	Error opening registry key
145	Recursiv Deletion of Registry Keys attempted
146	CodeBase: Code4Init not called
147	CodeBase: Code4Init error
148	CodeBase: error: can't load library
149	CodeBase warning: locking (r4locked)
150	The corresponding CodeBase database/object has been closed
151	SendKeys string error
152	SendKeys recursiv

Example

```
Local a$, i%
Debug.Show
For i% = 1 To 152
  Trace i%
  If Odd(i%) : Trace Err$(i%)
  Else : Trace Error$(i%)
  EndIf
Next i%
```

Returns the strings with GFA-BASIC error messages for codes 1 to 152.

See Also

Err Object, SysErr

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

SysErr Function

Purpose

Returns error message strings for the system error codes returned by **Err**.**LastDLLError**.

Syntax

$ = **SysErr**[$](error)

error:win32 error number

Description

SysErr$ returns the message string for an operating system error number.

Err$ returns the message string for a GFA-BASIC 32 error.

Example

```
OpenW 1
// error text for the error no. 3 of the
// used operating system
Print SysErr(3)
// error text for GFA-BASIC 32 error 3
Print Err$(3)
```

Remarks

Only part of the system errors have corresponding message strings.

See Also

Err Object, Err$

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Resume Command

Purpose

Resumes execution after an error-handling routine is finished.

Syntax

Resume [**Next | 0 |** *label*]

Description

The **Resume** statements can only be used in an error-handling routine defined with **On Error GoTo.**

The **Resume** or **Resume 0** are identical and (should) re-execute the line that caused the error. The **Resume** [**0**] command is useful when the error trap can fix the error situation. The program may retry to execute that line again and might continue without errors. However, **Resume** [**0**] doesn't work and generates an exception.

Resume Next (should) resume executing with the line immediately following the line that caused the error. However, **Resume Next** doesn't work and generates an exception. **Resume Next** command is only meaningful with **On Error Resume Next**.

The only working **Resume** statement is **Resume** *label*. Execution resumes at the label specified in the required argument. The label argument is a line label or line number and must be in the same procedure as the error handler. Actually, this is nothing else than **GoTo** *label*. The only difference is that **Resume** *label* re-initializes the **On Error** trap. Any new error following the label is catched in the same error trap, which might cause an infinite loop when an error occurs.

Inside the error trap the **On Error** mechanism is disabled.

Example

```
ResumeStatementDemo()
Close # 1
Kill "TESTFILE"

Sub ResumeStatementDemo()
  On Error GoTo errtrap
  Open "TESTFILE" for Output As # 1  ' Open file
    for output.
  Kill "TESTFILE"    ' Attempt to delete open file.
  labelx:
  Exit Sub
  errtrap:
  MsgBox Err.Number & Err.Description
  Resume labelx
End Sub
```

Remarks

A **Resume** [**Next** | **0**] command instructs the compiler to create code to hold the current executing line (4 bytes per line for subroutines smaller than 250 lines, and 7 bytes for larger routines). The code to maintain the position is generated between **On Error GoTo** *label* and **On Error GoTo 0** or the error trap staring with *label*. It seems the compiler generates faulty code for this process and halts with an exception.

On Error Resume Next instructs the compiler to generate optimized **Try**/**Catch** code around each code line (8 bytes extra per line). To prevent code bloat, you better use **Try**/**Catch**.

See Also

On Error, Try

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Bound Function

Purpose

Bounds test.

Syntax

n = **Bound**(n, lo, hi)

n, lo, hi:iexp

Description

The **Bound**(n, lo, hi) function tests whether the parameter n lies within the bounds of lo and hi (inclusive). This means that when n < lo or n > hi an error message is reported. Otherwise n is returned unchanged.

Example

```
OpenW # 1
Local i%, q%
Dim a%(49)
For i% = 1 To 20
  q% = Rand(49) + 1
  While a%(q%)
    q%++
  Wend
  Inc a%(q%)
Next i%
CloseW # 1
```

This programs selects 20 random numbers between 1 and 49 without repetition. The frequency of the number (zero or once) is noted in array a%().If Rand() returns a number for the second time the next higher number is taken instead. After many test runs an error (array index too big) appears several times.

To locate this error the line q%++ can, for example, be changed to

```
q% = Bound(q% + 1, 1, 49)
```

This will cause an error (Bound Error) on the line where q% is modified (q%++). In this way the place where the range is exceeded is easier to find.

Remarks

The **Bound**() function serves to find program errors by early discovery of any range violations.

See Also

BoundB(), BoundW(), BoundC()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

BoundB Function

Purpose

bounds test

Syntax

byte = **BoundB**(n)

byte = **BoundByte**(n)

n: integer expression

Description

The **BoundB**(n) function tests if the parameter n fits in a Byte. This means that when n < 0 or n > 255 an error message is reported. Otherwise n is returned unchanged.

Example

```
Local a| = 5, b| = 45, c|
c| = BoundB(a| * b|) //  5 * 45 = 225 - No Error
c| = BoundB(c| * 2)  // 225 * 2 = 450 - Array
 Bounds Error
// or...
c| = BoundByte(c| * 2)
```

Remarks

The **BoundB**() function serves to find program errors by early discovery of any range violations. **BoundByte** is a synonym.

See Also

Bound(), BoundW(), BoundC()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

BoundC Function

Purpose

bounds test

Syntax

Card = **BoundC**(n)

Card = **BoundCard**(n)

n: integer expression

Description

The **BoundC**(n) function tests if the parameter n fits in an unsigned word (Card). This means that when n < 0 or n > 65535 an error message is reported. Otherwise n is returned unchanged.

Example

```
Local a&, b% = 20000, addr% = V:a&
DPoke V:a&, BoundC(b%) // Checks that b% will fit
  in a Card/Word
Print a&               // Prints 20000
b% = 212000
DPoke V:a&, b%         // Not checking size of b%
  leads to...
Print a&               // ... a& = 15392 as only
  first 16 bits passed
DPoke V:a&, BoundC(b%) // This will flag up the
  error
```

```
// or simply..
~BoundCard(b%)
```

Remarks

The **BoundC**() function serves to find program errors by early discovery of any range violations. **BoundCard** is a synonym.

See Also

BoundB(), BoundW(), Bound()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

BackColor, ForeColor Properties

Purpose

BackColor returns or sets the background color of an Ocx object. **ForeColor** returns or sets foreground color used to display text and graphics in an Ocx object.

Syntax

[Object.]**BackColor** [= rgb]

[Object.]**ForeColor** [= rgb]

Object:Ocx Object
rgb:ivar

Description

When used without an object, the **BackColor** and **ForeColor** properties set the colors of the current active form object (**Form**, **LoadForm**, **Dialog**, **and OpenW**). The current active form is the one that is stored in **Me**. **Me** is set automatically after creating a form or by explicitly invoking **Set Me** = form Object.

As an alternative the colors can be set using the form properties .**BackColor** and .**ForeColor**.

For all forms and controls, the default settings are **BackColor = colBtnFace** and **ForeColor** = **colWindowText**.

Note: When using Common Controls version 6, it is NOT possible to change the text colour by setting the **ForeColor** property of any object of the 'Button' family: these include **CheckBoxes**, **Command** buttons, **Frames** and **Option** Boxes. In the case of **Command** buttons, this is also not possible using Common Controls version 5.

Example

```
OpenW # 1                        // Me = Win_1
ForeColor = QBColor(3)           // refers to Me
  implicitly
ForeColor = &H808080             // refers to Me
  implicitly
Me.ForeColor = RGB(92, 92, 92)   // use Me
  explicitly
Win_1.ForeColor = colBtnFace     // use the form's
  name
```

There are several methods to define the RGB color value for the form. The **RGB**()function is one way to define colors, and the **QBColor** function another. In most cases, it’s much easier to enter these numbers in hexadecimal.

The valid range for a normal RGB color is 0 to 16,777,215 ($FFFFFF). Each color setting (property or argument) is a 4-byte integer. The high byte of a number in this range equals 0. The lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 ($FF).

Consequently, you can specify a color as a hexadecimal number using this syntax: **$***BBGGRR.* The *BB* specifies the amount of blue, *GG* the amount of green, and *RR* the amount of red. Each of these fragments is a two-digit hexadecimal number from 00 to FF. The median value is 80.

Thus, the following number specifies gray, which has the median amount of all three colors: $808080

For RGB colors, the high byte equals 0 whereas for system colors the high byte equals 8. Setting the most significant bit to 1 changes the meaning of the color value: It no longer represents an RGB color, but an environment-wide color specified through the Windows Control Panel. The values that correspond to these system-wide colors range from &H80000000 to &H80000015. For example, the hexadecimal number used to represent the color of an active window caption is &H80000002. The following constants are predefined:

colScrollBar; colBackGround; colDesktop; colActiveCaption; colInactiveCaption; colMenu; colWindow; colWindowFrame colMenuText; colWindowText; colCaptionText; colActiveBorder; colInactiveBorder; colAppWorkSpace; colHighLight; colHighLightText; col3DFace; col3DShadow; colBtnFace; colBtnShadow; colGrayText; colBtnText; colInactiveCaptionText; colBtnHighLight; colBtnHiLight; col3DHighLight; col3DHiLight; col3DDkShadow; col3DLight; colInfoText; colInfoBk

These color constants define system colors that are recognized by the system by the high order byte value ($80). The translation to a RGB color value happens at system level. A property set to a system color constant remains having the index value! See also **GetRValue**().

Example

```
OpenW 1
BackColor = &H80000007
Ocx ListBox lb = , 10, 10, 100, 100
```

```
lb.AddItem "Text 1"
.BackColor = colAppWorkSpace
.ForeColor = _minInt + COLOR_HIGHLIGHTTEXT
Do
  Sleep
Until Me Is Nothing
```

Remarks

As an alternative for **BackColor** and **ForeColor** for forms you can use the **Color**, **RGBColor,** or **QBColor** commands. The [**RGB**]**Color** command takes RGB values (contrary to GFA-BASIC 16).

```
Color RGB(255, 0, 0), RGB(99, 99, 99)
```

If you set the **BackColor** property on a **Form** object, all text, and graphics, including the persistent graphics, are erased. This does not happen if you use the other color commands. Setting the **ForeColor** property doesn't affect graphics or print output already drawn. On all other controls, the screen color changes immediately.

See Also

Form, Color, RGBColor, QBColor, GetBValue

{Created by Sjouke Hamstra; Last updated: 03/03/2018 by James Gaite}

BkColor Property

Purpose

Returns or sets the background color for graphic commands.

Syntax

object.**BkColor** = [*value*]

object:Form Object
valueiexp

Description

Sets the background color for graphic commands. If you set the **BackColor** property on a **Form** object, all text, and graphics, including the persistent graphics (AutoRedraw), are erased. **BkColor** only sets the color, but doesn't erase the client area.

Initially, **BkColor** and **BackColor** have the same value.

Example

```
Form test
AutoRedraw = 1
Print "Backcolor: "; Hex(BackColor)
Print "BkColor: "; Hex(.BkColor)
DefFill 9
PBox 10, 35, 100, 125
.BkColor = RGB(0, 255, 255)
Text 0, 135, "New BkColor: " & Hex(.BkColor)
```

```
PBox 10, 150, 100, 240
Do
  Sleep
Until Me Is Nothing
```

Remarks

ForeColor sets the foreground color.

See Also

ForeColor, BackColor

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

RGB Function

Purpose

Returns a single color value from a set of red, green, and blue color components.

Syntax

x% = **RGB**(r, g, b)

x% : iexp
r, g, b : iexp

Description

Specifies the intensity of the red, green, and blue color components. The values can range from 0 to 255. Zero is the minimum color intensity; 255 is the maximum colour intensity.

RGB doesn't perform overflow checking. For instance, the value 256 is converted to 1.

Example

```
OpenW 1
Local col%
Line 10, 10, 10, 150
Auto
col% = RGB(150, 150, 150)
Color col%
Circle 30, 30, 100
Color RGB(-3, 510, -10)
```

```
Circle 100, 100, 150
```

Remarks

The other function to create a RGB color value _**RGB**() clips the passed values to the range 0 .. 255. Wrong values are corrected automatically. For instance, the value 257 is set to 255, and for negative values the colour value is rounded to zero.

RGB() is a bit faster, but doesn't perform overflow checking. Incrementing the color value will not result in an end color of white (255, 255, 255) like **_RGB**().

Another way to create the RGB value is by using the function **MakeL3L**().

See Also

_RGB, RGBColor, Color, RGBPoint

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

_RGB Function

Purpose

Returns a single color value from a set of red, green, and blue color components.

Syntax

x% = _**RGB**(r, g, b)

x% : iexp
r, g, b : iexp

Description

Specifies the intensity of the red, green, and blue color components. The values can range from 0 to 255. Zero is the minimum color intensity; 255 is the maximum color intensity.

_**RGB**() clips the passed values to the range 0 .. 255. Wrong values are corrected automatically. For instance, the value 257 is set to 255, and for negative values the color value is rounded to zero.

Example

```
OpenW 1
Local a%, col%
Line 10, 10, 10, 150
Auto
col% = _RGB(150, 150, 150)
Color col%
```

```
Circle 30, 30, 100
col% = _RGB(-3, 510, -10)
Color col%
Circle 100, 100, 150
KeyGet a
CloseW 1
```

Remarks

_RGB(250 + 20, 100 + 20, 80 + 20) results in **RGB**(255, 120, 100), not **RGB**(14, 120, 100) [14 == (270 And 255)]. _RGB is implemented as an optimized library function; it is not in-lined due to its complexity. As an illustration, the following code is required (more or less):

```
Function RGBAdd(ByVal Rgb1 As Int, ByVal hue As
  Int) As Int
  Dim tR As Int, tG As Int, tB As Int
  tR = GetRValue(Rgb1) + hue
  tG = GetGValue(Rgb1) + hue
  tB = GetBValue(Rgb1) + hue
  If tR > 255 Then tR = 255
  If tG > 255 Then tG = 255
  If tB > 255 Then tB = 255
  If tR < 0 Then tR = 0
  If tG < 0 Then tG = 0
  If tB < 0 Then tB = 0
  Return RGB(tR, tG, tB)
EndFunction
```

GFA-BASIC 32 brings it back to:

```
Function RGBAdd2(ByVal Rgb1 As Int, ByVal hue As
  Int) As Int
  Return _RGB(GetRValue(Rgb1) + hue,
    GetGValue(Rgb1) + hue, GetBValue(Rgb1) + hue)
EndFunction
```

By incrementing the r-g-b values using _RGB will eventually result in white (255,255,255).

The other function to create a RGB colour value **RGB**() is a bit faster, but doesn't perform overflow checking. For instance, the value 256 is converted to 1. Incrementing the colors using RGB() does not result in the end color white.

See Also

RGB, RGBColor, RGBPoint

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

GetBValue, GetGValue, GetRValue Function

Purpose

The **GetBValue, GetGValue**, and **GetRValue** function retrieves an intensity value for a color component of a 32-bit red, green, blue (RGB) value.

Syntax

Byte = **GetBValue**(rgb)

Byte = **GetGValue**(rgb)

Byte = **GetRValue**(rgb)

rgb: 32-bit RGB value

Description

The return value of **GetBValue** is the intensity of the blue component of the specified RGB color.

The return value of **GetGValue** is the intensity of the green component of the specified RGB color.

The return value of **GetRValue** is the intensity of the red component of the specified RGB color.

The intensity value is in the range 0 through 255.

Example

```
OpenW # 1
Local col%, nBlue%, nGreen%, nRed%, x%
// background color for a window
Win_1.BackColor = RGB(120, 250, 120)
// to get the whole color value
col% = Win_1.BackColor
// or for one pixel
// col% = GetPixel(Win_1.hDC , 380, 280)
Text 75, 10, "red"
Text 110, 10, "green"
Text 150, 10, "blue"
If col% > 0
  nRed% = GetRValue(col)
  nBlue% = GetBValue(col)
  nGreen% = GetGValue(col)
  Text 70, 40, nRed%
  Text 110, 40, nGreen%
  Text 150, 40, nBlue%
EndIf
```

Remarks

The **GetBValue, GetGValue**, and **GetRValue** functions are actually simple byte shift functions. Whatever you put in the parameter it will return. For example, when you assign a predefined color constant like **colBtnFace** (= $8000000F) you won't get the RGB-values of the color, but **GetBValue**(colBtnFace) = 0**, GetGValue**(colBtnFace) = 0, and **GetRValue**(colBtnFace) = $0F.

In addition, these functions do not work with the ARGB colours used with GDI+; to get the individual colour components you can use the **GetByte*n*()** functions as in the following example:

```
Local ARGB_Aquamarine = &HFF7FFFD4
```

```
Print Hex$(GetByte0(ARGB_Aquamarine))  // Alpha
 Value
Print Hex$(GetByte1(ARGB_Aquamarine))  // Red
 Value
Print Hex$(GetByte2(ARGB_Aquamarine))  // Green
 Value
Print Hex$(GetByte3(ARGB_Aquamarine))  // Blue
 Value
Print Hex$(GetRValue(ARGB_Aquamarine)) // Gets the
 Blue, not Red, Value
Print Hex$(GetBValue(ARGB_Aquamarine)) // Gets the
 Red, not Blue, Value
Print Hex$(GetGValue(ARGB_Aquamarine)) // Still
 gets the Green Value
```

See Also

GetByte0, GetByte1, GetByte2, GetByte3

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

SysCol Function

Purpose

Returns the system color of a specific element.

Syntax

c% = **SysCol**(e%)

e%:integer expression

Description

The **SysCol**() function returns the RGB value for the element specified in e%.

COLOR_ACTIVEBORDER(10) - active window

COLOR_ACTIVECAPTION(2) - active window caption

COLOR_APPWORKSPACE(12) - background of "multiple

COLOR_BACKGROUND(1) - desktop

COLOR_BTNFACE(15) - button surface

COLOR_BTNSHADOW(16) - button shadow

COLOR_BTNTEXT(18) - button text

COLOR_CAPTIONTEXT(9) - caption text

COLOR_GRAYTEXT(17) - gray (inactive) text field

COLOR_HIGHLIGHT(13) - selected items

COLOR_HIGHLIGHTTEXT(14) - text in selected items

COLOR_INACTIVATEBORDER(11) - inactive window frame

COLOR_INACTIVATECAPTION(3) - inactive caption

COLOR_MENU(4) - menu background color

COLOR_MENUTEXT(7) - menu text

COLOR_SCROLLBAR(0) - gray area in scroll bars

COLOR_WINDOW(5) - window background

COLOR_WINDOWFRAME(6) - window frames

COLOR_WINDOWTEXT(8) - color of text in windows

Example

```
OpenW 1
// to open a windows with the same
// background color as the surface color
// of the push button (Command)
Win_1.BackColor = SysCol(COLOR_BTNFACE)
Win_1.BackColor = GetSysColor(COLOR_BTNFACE)
' alternative, more conform MS Windows:
Win_1.BackColor = colBtnFace
Win_1.BackColor = &H8000000F
Win_1.BackColor = _minInt + COLOR_BTNFACE
```

The following code shows the system colours as they are manifested on your system,

```
OpenW Full 1
Global Int colour = $80000000, n, y
```

```
Local a$
For n = 0 To 24
  Color colour + n
  PBox 10, y, 20, y + 10
  Color 0
  Read a$ : Text 25, y, a$ & ": " & Hex(Point(11, y
    + 1), 6) &  "  "
  Add y, 20
Next n
Data
  "Scrollbars","Desktop","ActiveTitleBar","Inactive
  TitleBar","MenuBar"
Data
  "WindowBackground","WindowFrame","MenuText","Wind
  owText","TitleBarText"
Data
  "ActiveBorder","InactiveBorder","ApplicationWorkS
  pace","Highlight","HighlightText"
Data
  "ButtonFace","ButtonShadow","GrayText","ButtonTex
  t","InactiveCaptionText"
Data
  "3DHighlight","3DDKShadow","3DLight","InfoText","
  InfoBackground"
```

Remarks

SysCol is short for the Windows API function *GetSysColor*().

See Also

Color, RGBColor, BkColor, ForeColor, BackColor

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Cls Command

Purpose

Clears the screen.

Syntax

Cls [color]

Description

Deletes the contents of the actual window. The window is deleted with the background color set with **BackColor**.

When **AutoRedraw** is used and the argument *color* is specified, a VGA color is used when color is in the range 0..15. Other wise the color is interpreted as RGB value.

Example

```
Local a%
OpenW # 1
AutoRedraw = 1
Print "Press any key"
KeyGet a%
Cls 5
// Cls doesn't reset BackColor or BkColor
Win_1.BkColor = QBColor(5)
Print "Press any key"
KeyGet a%
CloseW 1
```

See Also

BackColor

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

DefLine Command

Purpose

defines the line type.

Syntax

DefLine [style] [, thickness]

style, thickness:integer expression

Description

DefLine defines the appearance of a line drawn using the **Line**, **Box**, **RBox**, **Circle**, **Ellipse** and **Polyline** commands.

The first parameter determines the appearance of the line as follows:

style = 0 or PS_SOLID	solid line
style = 1 or PS_DASH	dashed line
style = 2 or PS_DOT	dotted line
style = 3 or PS_DASHDOT	dash-dot line
style = 4 or PS_DASHDOTDOT	-..-..- line
style = 5 or PS_NULL	invisible border
style = 6 or PS_INSIDEFRAME	dithered, e.g., color emulation
style = 7 or PS_USERSTYLE	an array with user defined styles.
style = 8 or PS_ALTERNATE	each other pixel will be set, only useable with

	PS_COSMETIC
style = 15 or PS_STYLE_MASK	can have one of the styles above
style = 0 or PS_ENDCAP_ROUND	end of the line will be rounded
style = 256 or PS_ENDCAP_SQUARE	end of the line will be square
style = 512 or PS_ENDCAP_FLAT	end of the line is flat
style = 3840 or PS_ENDCAP_MASK	can get one value of the three possible (0, 256, 512)
style = 0 or PS_JOIN_ROUND	join is round
style = 4096 or PS_JOIN_BEVEL	join is bevel
style = 8192 or PS_JOIN_MITER	join is miter
style = 0xF000 or PS_JOIN_MASK	can get one of the three possible values (0, 4096, 8192)
style = 0 or PS_COSMETIC	fixed width and fixed height of a used line, very quick
style = 0x10000 or PS_GEOMETRIC	scaleable line with fixed are variable style, and with the width of more as one pixel
style = 0xF0000 or PS_TYPE_MASK	can contents PS_COSMETIC or PS_GEOMETRIC

thickness specifies the line thickness in pixels.

Warning! When thickness is over 1, a solid line is always drawn.

Example

Example 1

```
OpenW 1
Local a%, i%
Color RGB(0, 255, 0)
For i% = 0 To 4
  DefLine i%, 1
  Line 0, (i% + 1) * _Y / 6, _X, (i% + 1) * _Y / 6
Next
KeyGet a% // Press any key
For i% = 0 To 4
  DefLine i%, i%
  Line 0, (i% + 1) * _Y / 6, _X, (i% + 1) * _Y / 6
Next
Do : Sleep : Until Win_1 Is Nothing
```

Example 2

```
OpenW 1
Local i&, j&, stp&
DefLine PS_INSIDEFRAME, 99
stp& = 20
For i& = 0 To _X Step stp&
  j& = i& * 255 / _X
  RGBColor RGB(255 - j&, 0, j&)
  PBox i&, 0, i& + stp& - 1, _Y
Next i&
Do : Sleep : Until Win_1 Is Nothing
```

Remarks

DefLine internally uses the Windows function *CreatePen*(). The line color must be set beforehand with **BkColor**, **Color, RGBColor,** or **QBColor**.

See Also

DefFill

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

DefFill Command

Purpose

defines a fill pattern.

Syntax

DefFill pattern

DefFill p$

pattern: integer expression
p$: string

Description

DefFill *pattern* defines a fill pattern for **PBox**, **PCircle**, **PEllipse**, **Polyfill** and **Fill** graphic commands. One the 48 available dot or line patterns can be selected using the pattern option. (see Fill pattern table).

DefFill p$ defines a custom monochrome fill pattern. The string is 8 bytes long, where each byte specifies the 8-bits pattern for a row. Together the 8 bytes define a 8 x 8 bit pattern.

Example

// Fill pattern table

```
Local h%, i%, j%, w%, ye%, ys%
OpenW # 1
w% = _X / 12, h% = _Y / 4
```

```
For i% = 1 To 48
  Switch i%
  Case To 12
    j% = i%
    ys% = 0, ye% = _Y / 4
  Case 13 To 24
    j% = Sub(i%, 12)
    ys% = _Y / 4, ye% = _Y / 2
  Case 25 To 36
    j% = Sub(i%, 24)
    ys% = _Y / 2, ye% = _Y * 3 / 4
  Case 37 To 48
    j% = Sub(i%, 36)
    ys% = _Y * 3 / 4, ye% = _Y
  EndSwitch
  DefFill i%
  PBox (j% - 1) * w%, ys%, (j% - 1) * w% + w%, ye%
Next i%
```

draws 48 rectangles using various fill patterns.

```
Local x$ = Chr$(0, $FF, 0, $FF, 0, $FF, 0, $FF)
DefFill x$
PBox 8, 8, 100, 100
```

See Also

DefLine

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

DrawMode Property

Purpose

Returns or sets a value that determines the appearance of output.

Syntax

[object.]**DrawMode** [= n]

object:Form or Printer object
n:iexp

Description

Use this property to produce visual effects with the graphic output commands (**Line**, **Circle**, etc). Each pixel in the draw pattern is compared to the corresponding pixel in the existing background and then applies bit-wise operations.

R2_BLACK	points are always black.
R2_WHITE	sets white points.
R2_NOP	points are not changed.
R2_NOT	point corresponds to the inverse of the screen color.
R2_COPYPEN	color set with **Color**.
R2_NOTCOPYPEN	inverse of color in Color
R2_MERGEPENNOT	set point corresponds to the pen color "or-ed" with the inverse screen color.
R2_MASKPENNOT	set points corresponds to pen color "and-ed" with inverse pen color.

R2_MERGENOTPEN	set point corresponds to screen color "and-ed" with the inverse pen color.
R2_MERGEPEN	point color corresponds to the pen color "or-ed" with the screen color.
R2_NOTMERGEPEN	inverse R2-MERGEPEN color.
R2_MASKPEN	point corresponds to colors in screen and pen (logical And).
R2_NOTMASKPEN	point corresponds to inverse R2-MASKPEN color.

Using **DrawMode** without an object will affect the current active output object, usually **Me** (unless Output = Printer is used).

DrawMode is a get/put property and can be read as well.

Example

```
OpenW 1
Local a%
RGBColor RGB(125, 125, 125), RGB(150, 100, 150)
DefFill 8
PBox 10, 10, 100, 200
// Graphmode 1 is Default
PBox 15, 15, 105, 205
KeyGet a%
// waiting of a key
Cls
DrawMode = R2_MERGEPENNOT // Or operation
PBox 10, 10, 100, 200
PBox 15, 15, 105, 205
KeyGet a%
// waiting of a key
Cls
DrawMode = R2_XORPEN // Xor op
PBox 10, 10, 100, 200
PBox 15, 15, 105, 205
KeyGet a%
// waiting of a key
Cls
DrawMode = R2_MASKPEN // And op
PBox 10, 10, 100, 200
PBox 15, 15, 105, 205
KeyGet a%
// waiting of a key
CloseW 1
```

Draws two rectangular, one over the other.

Remarks

DrawMode is the VB compatible implementation of the GFA-BASIC **GraphMode** command. In addition, **DrawMode** is a property.

See Also

GraphMode

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

GraphMode Command

Purpose

Control of graphic output on bit level

Syntax

GraphMode n [,m]

Description

GraphMode n defines the relationship between the graphic output and the screen. This relationship involves the bit-wise combination of the current screen contents and the new graphic which is to be drawn. The parameter n specifies how this combination is to be performed. Following modes are possible:

R2_BLACK	points are always black.
R2_WHITE	sets white points.
R2_NOP	points are not changed.
R2_NOT	point corresponds to the inverse of the screen color.
R2_COPYPEN	color set with Color.
R2_NOTCOPYPEN	inverse of color in Color
R2_MERGEPENNOT	set point corresponds to the pen color "or-ed" with the inverse screen color.
R2_MASKPENNOT	set points corresponds to pen color "and-ed" with inverse pen color.
R2_MERGENOTPEN	set point corresponds to screen color

	"and-ed" with the inverse pen color.
R2_MERGEPEN	point color corresponds to the pen color "or-ed" with the screen color.
R2_NOTMERGEPEN	inverse R2-MERGEPEN color
R2_MASKPEN	point corresponds to colors in screen and pen (logical And).
R2_NOTMASKPEN	point corresponds to inverse R2-MASKPEN color.
R2_XORPEN	set point is either in screen color or pen color but not in both (logical Xor).
R2_NOTXORPEN	point color corresponds to the inverse R2_XORPEN color.

GraphMode 1 (R2_BLACK) is default.

The second optional parameter **GraphMode ,**m can take on the values **OPAQUE** and **TRANSPARENT**. OPAQUE overwrites the background and is the default.

Example

```
Dim a%
OpenW # 1
DefFill 4
PBox 10, 10, 100, 200     //Graphmode 1 default
PBox 15, 15, 105, 205
Delay 1
Cls
GraphMode R2_MERGEPEN     //logical Or
PBox 10, 10, 100, 200
PBox 15, 15, 105, 205
Delay 1
PBox 10, 10, 100, 200
PBox 15, 15, 105, 205
```

```
Delay 1
Cls
GraphMode R2_MASKPEN      //logical And
PBox 10, 10, 100, 200
PBox 15, 15, 105, 205
Delay 1
CloseW # 1
```

draws two overlapping rectangles.

Remarks

GraphMode n conforms to the **DrawMode** property of the window/form.

GraphMode ,m conforms to the **FontTransparent** property of the window/form.

See Also

DrawMode, FontTransparent

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Print Command

Purpose

Prints text into the current active **Form** or **Printer** object.

Syntax

Print x[,y,a$,...][;]

x,y:aexp
a$:sexp

Description

A **Print** without any parameters performs a line feed. If **PrintScroll** = 1 and the cursor is on the last line, the whole screen is moved up by one line. A **Print** followed by an expression prints this expression at the current cursor position.

Print At, **Locate**, **VTab,** and **HTab** can be used to position the cursor. The strings must be enclosed in quotation marks. **Print** can be followed by several (different) expressions which must be separated by a comma, a semi-colon, or an apostrophe.

The comma moves the cursor to the next tab position - a column fully divisible by 16. When the last column is reached the cursor is moved to column 17 on the next line. The semi-colon performs the output of expressions without any spaces between them. The apostrophe, however, inserts a space between the expressions. A line feed is performed after each **Print** except when the last expression is followed by a semi-colon. In such a case the next **Print** output resumes from the end of the previous one.

All data printed is formatted using the decimal separator according the **Mode Using** setting. Use the **Using** function to format the output before printing.

For Boolean data, either True or False is printed. The **True** and **False** keywords are translated according to the locale setting for the host application.

A Date is written according the **Mode Date** setting.

Example

```
OpenW 1
Local a$, b$
Print 3 * 4 + 12
Print "3 * 4 + 12 = "; 3 * 4 + 12
a$ = "GFA Software Technologies"
b$ = "-BASIC 32"
Print Left$(a$, 3) + b$
Print "A"``Chr$(66)``"C"
Print "a$,b$: "; a$, b$
```

Remarks

Because the **Print** method typically prints with proportionally-spaced characters, there is no correlation between the number of characters printed and the number of fixed-width columns those characters occupy. For example, a wide letter, such as a "W", occupies more than one fixed-width column, and a narrow letter, such as an "i", occupies less. To allow for cases where wider than average characters are used, your tabular columns must be positioned far enough apart. Alternatively, you can print using a fixed-pitch font (such as Courier) to ensure that each character uses only one column. Use the **Font** object to adjust the font settings.

See Also

PrintAt, Using, Write, Text, Mode

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Locate, LocaXY and LocaYX Commands

Purpose

Cursor positioning

Syntax

Locate row, column

LocaXY column, row

LocaYX row, column

row, column:ivar

Description

Places the cursor at position x (column) and y (row). The exact location depends on the size of the font selected in the Form.

Example

```
OpenW # 1
Print "Hello GFA"
Locate 12, 4
Print "Hello GFA with Locate"
LocaXY 15, 8
Print "Hello GFA with LocaXY"
LocaYX 15, 8
Print "Hello GFA with LocaYX"
```

Remarks

Print AT() combines the functions of **Locate** and the subsequent **Print** commands.

See Also

Print At, VTab, HTab, LocaXY, LocaYX

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

Print At Commands

Purpose

Prints text at a specific position on the current active **Form** or **Printer** object.

Syntax

Print At(column,row);exp1 [**,**[**At**(column,row;] exp2,...]

Print ATXY(column, row);exp[**,**[**ATXY**(column, row);]exp2,...]

Print ATYX(row, column);exp[**,**[**ATYX**(row, column);]exp2,...]

column, row:iexp, cursor position
exp1, exp2:aexp or sexp

Description

Print At(column, row) followed by an expression, performs the output of this expression at the cursor position defined by column and row. **Print At**() without any parameters performs a line feed. The list of parameters after **Print At**() can contain other **At**() instructions which then apply to printing of expressions following after them. i.e. at the corresponding column and row.

Print ATXY(column, row) is the same as **Print At**(column, row) and **Print ATYX**(row, column) different only in the order of the parameters - it states the row first, not the column.

Example

```
Local a%
OpenW # 1
Print AT(7, 12); "What do you get";
Print AT(7, 13); "when you multiply"
Print AT(7, 14); "6 by 7"; AT(7, 16); " 42!!! "
Print ATXY(4, 6); "What do you get";
Print ATXY(4, 7); "when you multiply"
Print ATXY(4, 8); "6 by 8"; ATXY(4, 10); " 48!!! "
Print ATYX(1, 1); "What do you get";
Print ATYX(2, 1); "when you multiply"
Print ATYX(3, 1); "6 by 9"; ATYX(4, 1); " 54!!! "
```

Remarks

The **Text** command is recommended for output. It is considerably faster.

See Also

Print, Mode, Text

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

HTab, VTab Commands

Purpose

Positions the cursor positioning in either the vertical or horizontal planes.

Syntax

HTab column

VTab row

column, row:integer expression

Description

Places the cursor in the column or row specified in respective integer variable.

Example

```
OpenW # 1
Print AT(1, 1); "Hello GFA"
HTab 20
VTab 20
Print "Hello GFA"
```

Prints Hello GFA from the first column on the first line, and then prints the same string again only from the 20th row and 20th column.

The same:

```
Print AT(20, 20); "Hello GFA"
```

See Also

Locate, PrintAt, Tab

{Created by Sjouke Hamstra; Last updated: 01/03/2017 by James Gaite}

DrawText Command

Purpose

Displays formatted text.

Syntax

DrawText x1, y1, x2, y2, t$, mode

x1, y1, x2, y2:floating-point exp
t$:sexp
mode:iexp

Description

DrawText works in principle like **Text**; however the text can be formatted by using the last parameter mode. It must be taken into account that Windows can clip text output to a rectangle. This occurs for example in multi-line Combo boxes. The formatted output is therefore limited to a rectangular area whose height (in case of single line text) is determined by the font height. The text specified in t$ is displayed at the output coordinates x and y. mode can assume the following values for a formatting with a logical Or:

DT_BOTTOM ($0008)	draws a single line of text at the bottom of a rectangular area. This only works with single line text and must have the DT_SINGLELINE mode specified as well.
DT_CALCRECT ($0400)	determines the width and height

	of a rectangular area.
DT_CENTER ($0001)	centers text within a rectangular area.
DT_EXPANDTABS ($0040)	expands the tab stops.
DT_EXTERNALLEADING ($0200)	expands the height of a text line by the distance between two lines.
DT_LEFT($0000)	draws text left justified.
DT_NOCLIP ($0100)	turns the clipping to a rectangular area off.
DT_NOPREFIX ($0800)	disables the default function of the "&" character (display the following characters as underlined.
DT_RIGHT ($0002)	draws text right justified.
DT_SINGLELINE ($0020)	specifies a single line of text.
DT_TABSTOP ($0080)	sets tab stops. The high byte of attr% contains the number of characters per tab.
DT_TOP ($0000)	draws a single line of text at the top edge of a rectangular area.
DT_VCENTER ($0004)	displays a single line of text vertically centered. This only works with single line text and must have the DT_SINGLELINE mode specified as well.
DT_WORDBREAK ($0010)	turns on word wrap.
DT_EDITCONTROL	Duplicates the characteristics of a multi line edit control
DT_PATH_ELLIPSIS or	Replaces part of the given string

DT_END_ELLIPSIS	with ellipses, if necessary, so that the result fits in the specified rectangle. The given string is not modified unless the DT_MODIFYSTRING flag is specified.
DT_MODIFYSTRING	Modifies the given string to match the displayed text. This flag has no effect unless the DT_END_ELLIPSIS or DT_PATH_ELLIPSIS flag is specified.
DT_RTLREADING	Layout in right to left reading order for bi-directional text when the font selected into the *hDC* is a Hebrew or Arabic font. The default reading order for all text is left to right.
DT_WORD_ELLIPSIS	Truncates text that does not fit in the rectangle and adds ellipses.

Note The DT_CALCRECT, DT_EXTERNALLEADING, DT_INTERNAL, DT_NOCLIP, and DT_NOPREFIX values cannot be used with the DT_TABSTOP value.

Example

```
OpenW # 1
Local a$ = "Hello" + Chr$(13) + "Bye..."
DrawText 10, 20, 110, 120, a$, DT_NOCLIP |
  DT_WORDBREAK
```

Prints "Hello" and then on the next line "Bye", ignoring the clipping rectangle.

Remarks

DrawText corresponds to Windows function *DrawText*.

See Also

Text, GrayText

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

Text Command

Purpose

Output of an expression as graphic text

Syntax

Text x!, y!, sexp

x!, y!:Single
sexp:svar or sexp

Description

Text x, y, sexp prints expression exp as graphic text at coordinates x, y. The **ScaleMode** property determines the unit of measure used. The point defined with x, y is aligned with the left corner of the base line of the first character in *exp*. The color of the text is set using **RGBColor**, **Color**, or **QBColor**. When **FontTransparent** = True or **GraphMode** ,TRANSPARENT the background is not overwritten with the the background color (2nd parameter of (**RGB**)**Color** and **QBColor**. Otherwise the background of the text is filled.

Example

```
OpenW # 1
Dim i%, s$ = "Test Test Test"
For i% = 0 To 10
  Text 50, Add(Shl(i%, 4), 16), s$
Next i%
```

Writes "Test Test Test" in different positions down the screen.

Remarks

Text is considerably faster than **Print**.

How the text is aligned can be altered by the **SetTextAlign**() API as shown by the following example:

```
OpenW 1
// Default 'Top' text alignment
Text 60, 20, "Hello" : FontSize = 12 : Text 90,
  20, "Hello" : FontName = "Courier" : Text 135,
  20, "Hello"
Text 60, 40, "Hello"
// Outputting Text to align along the base line
~SetTextAlign(Win_1.hDC, 24)
' 24 = TextOut y-coordinate = baseline;
  SetTextAlign set to 0 or top by default
' SetTextAlign affects Text as the latter uses the
  TextOut API
FontName = "MS Shell Dlg" : FontSize = 8
Text 60, 80, "Hello" : FontSize = 12 : Text 90,
  80, "Hello" : FontName = "Courier" : Text 135,
  80, "Hello"
Text 60, 100, "Hello"
' Note: SetTextAlign also affects Print statements
  as follows:
Print "Line 1" // is printed above the top of the
  work area of the window
Print "Line 2"
Print "Line 3"
```

SetTextAlign() can be used with the following constants

TA_BASELINE = 24 - The reference point will be on the baseline of the text.

TA_BOTTOM = 8 - The reference point will be on the bottom edge of the bounding rectangle of the text.

TA_CENTER = 6 - The reference point will be horizontally centered along the bounding rectangle of the text.

TA_LEFT = 0 - The reference point will be on the left edge of the bounding rectangle of the text.

TA_NOUPDATECP = 0 - Do not set the current point to the reference point.

TA_RIGHT = 2 - The reference point will be on the right edge of the bounding rectangle of the text.

TA_RTLREADING = 256 - Win 95/98 only:Display the text right-to-left (if the font is designed for right-to-left reading).

TA_TOP = 0 - The reference point will be on the top edge of the bounding rectangle of the text.

TA_UPDATECP = 1 - Set the current point to the reference point.

See Also

Print, Print At, TextXor, GrayText, ScaleMode, Color, RGBColor, QBColor

{Created by Sjouke Hamstra; Last updated: 14/01/2015 by James Gaite}

TextXor Command

Purpose

Output of an expression as graphic text with a bitwise exclusive OR of the destination and source.

Syntax

TextXor x!,y!, exp

x!,y!:Single
exp:svar or sexp

Description

TextXor *x, y, exp* prints expression *exp* as graphic text at coordinates x, y. The **ScaleMode** property determines the unit of measure used. The foreground color of the text is set using **RGBColor**, **Color**, or **QBColor**.

TextXor allows to place text on the background without disturbing the background. Under Windows 3.1 often used construction **GraphMode** R2_XORPEN : Text x, y, exp is ignored under Windows 95. This makes it impossible to restore the background when the text is displayed twice in the R2_XORPEN grahpmode.

Example

```
OpenW 1
Dim x As Int, y As Int, k As Int
Dim xo As Int, yo As Int
For x = 0 To _X Step 40
```

```
  Line x, 0, x, _Y
Next x
For y = 0 To _Y Step 40
  Line 0, y, _X, y
Next y
y = -80, yo = y
Global doexit As Boolean = False
Do
  Sleep
  If !doexit
    Mouse x, y, k
    If x != xo || y != yo || k = 1
      TextXor xo, yo, xo & yo
      If k = 1 Then QBColor Rand(16) : _
        Circle x, y, 24
      xo = x : yo = y
      TextXor x, y, x & y
    EndIf
  EndIf
Until Me Is Nothing

Sub Win_1_Close(Cancel?)
  doexit = True
EndSub
```

Writes "Test Test Test" in different ways to the screen.

Remarks

GrayText is another variant on **Text**.

See Also

Print, Print At, Text, GrayText, ScaleMode, Color, RGBColor, QBColor

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

GrayText Command

Purpose

Displays given text in gray.

Syntax

GrayText x, y, t$

x, y:Single

Description

GrayText works in principle like **Text**, however, the string expression is shown in gray. As a rule Windows uses gray to indicate when an entry is not selectable.

The command requires three parameters. The first two x and y set the X and Y coordinates for the origin of the string specified in t$.

Example

```
OpenW 1
FontSize = 40
Text 10, 20, "Hello GFA"
GrayText 10, 40, "Hello GFA"
```

prints "Hello GFA", first in default color and then in gray.

See Also

Text, DrawText

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

RBox, PRBox Commands

Purpose

Draws a rectangle with rounded corners.

Syntax

RBox x1,y1,x2,y2
RBox x1,y1 **To** x2,y2
RBox x1,y1, **Step** w,h

PRBox x1,y1,x2,y2
PRBox x1,y1 **To** x2,y2
PRBox x1,y1, **Step** w,h

x1,y1,x2,y2,w,h : single exp

Description

RBox x1,y1,x2,y2 and **RBox** x1,y1 **To** x2,y2 both draw a rectangle with rounded corners, with the diagonally opposite corner coordinates at x1,y1 (upper left) and x2,y2 (lower right), while **RBox** x1,y1 **Step** w,h also draws a similar rectangle but with top left coordinate x1,y1 and a width of w and height of h.

The width of the line drawn depends on the setting of the **DefLine** command and the way a line or box is drawn on the background depends on the setting of the **DrawMode** and **BkColor** properties.

The **PRBox** command acts very much the same, except that the boxes drawn are filled with a pattern defined using

Deffill.

Example

```
OpenW 1
RBox 10, 10, 100, 100
DefLine 1
RBox 110, 10, Step 90, 90
PRBox 10, 110, 100, 200
DefFill 5 : DefLine  0
PRBox 110, 110, Step 90, 90
```

See Also

BkColor, DefFill, DefLine, DrawMode, Box, PBox, Box3D, PBox3D, PolyLine, PolyFill

{Created by Sjouke Hamstra; Last updated: 22/06/2017 by James Gaite}

Box3D, PBox3D Commands

Purpose

Draws a 3D rectangle

Syntax

Box3D x1, y1, x2, y2 [, [edge][,bf]]
Box3D x1, y1 **To** x2, y2 [, [edge][,bf]]
Box3D x1, y1, **Step** w, h [, [edge][,bf]]

PBox3D x1, y1, x2, y2 [, [edge][,bf]]
PBox3D x1, y1 **To** x2, y2 [, [edge][,bf]]
PBox3D x1, y1, **Step** w, h [, [edge][,bf]]

h, w, x1, x2, y1, y2	*: single*
edge	*: EDGE_ constants*
bf	*: BF_ constants*

Description

Box3D x1,y1,x2,y2 and Box3D x1, y1 **To** x2, y2 draw a 3D rectangle with diagonal corner coordinates x1,y1 (upper left) and x2,y2 (lower right), while B>Box3D x1,y1, **Step** x2,y2 draws a similar rectangle but with upper left coordinates x1,y1 and a width of w and a height of h. The optical effect is specified by using the constants *edge* and *bf* *(*default: edge= EDGE_RAISED and bf = BF_RECT).

The Edge constants come in three forms:

1. Those that affect the inner edge only:

BDR_RAISEDINNER ($4) - Draws a raised inner edge.
BDR_SUNKENINNER ($8) - Draws a sunken inner edge.

2. Those that affect the outer edge only:

   BDR_RAISEDOUTER ($1) - Draws a raised outer edge.
   BDR_SUNKENOUTER ($2) - Draws a sunken outer edge.

3. Those that affect the both edges:

   EDGE_BUMP ($9) - Combination of BDR_RAISEDOUTER and BDR_SUNKENINNER.
   EDGE_ETCHED ($6) - Combination of BDR_SUNKENOUTER and BDR_RAISEDINNER.
   EDGE_RAISED ($5) - Combination of BDR_RAISEDOUTER and BDR_RAISEDINNER.
   EDGE_SUNKEN ($A) - Combination of BDR_SUNKENOUTER and BDR_SUNKENINNER

The Border (BF) constants determine which borders are affected and are as follows:

BF_ADJUST ($2000) - Shrink the rectangle to exclude the edges that were drawn.
BF_BOTTOM ($0008) - Draw bottom of border rectangle only.
BF_BOTTOMLEFT ($0009) - Draw bottom and left side of border rectangle.
BF_BOTTOMRIGHT ($000A) - Draw bottom and right side of border rectangle.
BF_DIAGONAL ($0010) - Diagonal border.
BF_DIAGONAL_ENDBOTTOMLEFT ($0019) - Diagonal border. The end point is the lower-left corner of the rectangle; the origin is top-right corner.
BF_DIAGONAL_ENDBOTTOMRIGHT ($001A) -

Diagonal border. The end point is the lower-right corner of the rectangle; the origin is top-left corner.
BF_DIAGONAL_ENDTOPLEFT ($0013) - Diagonal border. The end point is the top-left corner of the rectangle; the origin is lower-right corner.
BF_DIAGONAL_ENDTOPRIGHT ($0016) - Diagonal border. The end point is the top-right corner of the rectangle; the origin is lower-left corner.
BF_FLAT ($4000) - Flat border.
BF_LEFT ($0001) - Left side of border rectangle.
BF_MIDDLE ($0800) - Interior of rectangle to be filled.
BF_MONO ($8000) - One-dimensional border.
BF_RECT ($000F) - Entire border rectangle.
BF_RIGHT ($0004) - Right side of border rectangle.
BF_SOFT ($1000) - Soft buttons instead of tiles.
BF_TOP ($0002) - Top of border rectangle.
BF_TOPLEFT ($0003) - Top and left side of border rectangle.
BF_TOPRIGHT ($0006) - Top and right side of border rectangle.

Example

```
OpenW 1, , , 370, 465
TitleW 1, "Example: GFA-BASIC 32 Border Box3D +
  PBox3D"
FontSize = 9
FontBold = True
Text 10, 5, "EDGE"
Text 10, 40, "EDGE_RAISED"
Text 10, 90, "EDGE_ETCHED"
Text 10, 140, "EDGE_BUMP"
Text 10, 190, "EDGE_SUNKEN"
Text 10, 240, "BDR_RAISEDOUTER"
Text 10, 290, "BDR_SUNKENOUTER"
Text 10, 340, "BDR_RAISEDINNER"
Text 10, 390, "BDR_SUNKENINNER"
Text 160, 5, "Box3D"
Box3D 160, 30, Step 40, 40// Default
Box3D 160, 80, Step 40, 40, EDGE_ETCHED
Box3D 160, 130, Step 40, 40, EDGE_BUMP
Box3D 160, 180, Step 40, 40, EDGE_SUNKEN
Box3D 160, 230, Step 40, 40, BDR_RAISEDOUTER
Box3D 160, 280, Step 40, 40, BDR_SUNKENOUTER
Box3D 160, 330, Step 40, 40, BDR_RAISEDINNER
Box3D 160, 380, Step 40, 40, BDR_SUNKENINNER
Box3D 160, 380, Step 40, 40, BDR_OUTER, BF_MIDDLE
Text 205, 5, "BF_SOFT"
Box3D 205, 30, Step 40, 40, EDGE_RAISED, BF_SOFT
Box3D 205, 80, Step 40, 40, EDGE_ETCHED, BF_SOFT
Box3D 205, 130, Step 40, 40, EDGE_BUMP, BF_SOFT
Box3D 205, 180, Step 40, 40, EDGE_SUNKEN, BF_SOFT
Box3D 205, 230, Step 40, 40, BDR_RAISEDOUTER,
  BF_SOFT
Box3D 205, 280, Step 40, 40, BDR_SUNKENOUTER,
  BF_SOFT
Box3D 205, 330, Step 40, 40, BDR_RAISEDINNER,
  BF_SOFT
Box3D 205, 380, Step 40, 40, BDR_SUNKENINNER,
  BF_SOFT
Text 280, 5, "PBox3D"
PBox3D 280, 30, Step 40, 40, EDGE_RAISED
PBox3D 280, 80, Step 40, 40, EDGE_ETCHED
PBox3D 280, 130, Step 40, 40, EDGE_BUMP
PBox3D 280, 180, Step 40, 40, EDGE_SUNKEN
PBox3D 280, 230, Step 40, 40, BDR_RAISEDOUTER
PBox3D 280, 280, Step 40, 40, BDR_SUNKENOUTER
PBox3D 280, 330, Step 40, 40, BDR_RAISEDINNER
PBox3D 280, 380, Step 40, 40, BDR_SUNKENINNER
Do
  Sleep
Until Me Is Nothing
```

Remarks

Box3D and **PBox3D** use the *DrawEdge* API function.

See Also

Box, PBox, RBox, PRBox, PolyLine, PolyFill

{Created by Sjouke Hamstra; Last updated: 22/06/2017 by James Gaite}

Circle, PCircle Commands

Purpose

Draws a circle.

Syntax

Circle x, y, r [,w1, w2]

PCircle x, y, r [,w1, w2]

x, y, r, w1, w2 : Single expression

Description

Circle x, y, r[,w1,w2] draws a circle with the radius r around the centre with the coordinates x and y. In addition, by using the start (w1) and end (w2) angles, you can draw just an arc rather than the full circle - the angles w1 and w2 are given in whole degree steps as per Figure 1, with any arc being drawn in an anti-clockwise direction.

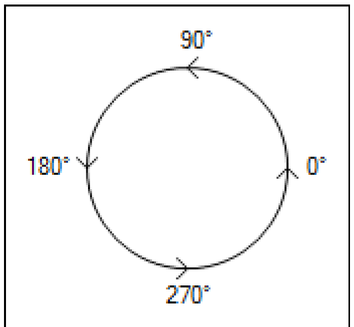


Figure 1

The width of the line drawn depends on the setting of the **DefLine** command, while the way a line or box is drawn on the background depends on the setting of the **DrawMode** and **BkColor** properties.

The **PCircle** command acts very much the same, except that the circles drawn are filled with a pattern defined using **Deffill**.

Example

```
OpenW 1
Circle 100, 100, 20, 90, 180  // Draws a quarter arc...
DefLine 0, 10
Circle 100, 100, 60           // ...inside a full circle.
DefLine 0, 1 : DefFill 5
PCircle 250, 100, 60          // Draws a filled circle...
DefLine 2 : DefFill 48
PCircle 250, 100, 60, 45, 90  // ...with a pie section.
```

Remarks

The current scaling depends of the form's **ScaleMode** setting.

The **Circle** and **PCircle** commands use the old GDI library. For a smoother circle drawn using anti-aliasing, you can use Windows GDI+ library instead.

Known Issues

Note: When the radius *r* is declared as a **Byte** or **Short**/**Word** the circle isn't drawn; this can be got around by using **CSng**(*r*). **Double**, **Int32** and **Int64** variables are unaffected.

See Also

Ellipse, PEllipse, ScaleMode

{Created by Sjouke Hamstra; Last updated: 17/12/2015 by James Gaite}

Ellipse, PEllipse Commands

Purpose

Draws an ellipse.

Syntax

Ellipse x, y, rx, ry [,w1, w2]

PEllipse x, y, rx, ry [,w1, w2]

x, y, rx, ry, w1, w2 : single exp

Description

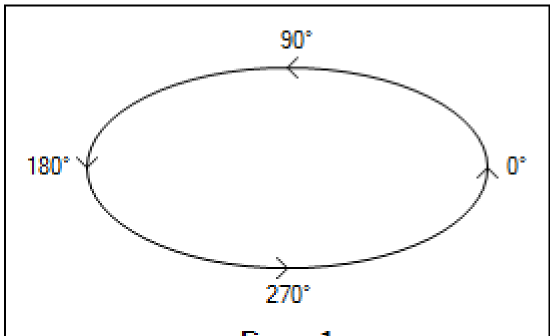


Figure 1

Ellipse x, y, rx, ry[,w1,w2] draws an ellipse with the horizontal radius rx and the vertical radius ry, around the centre point with coordinates x and y. In addition, by using the start (w1) and end (w2) angles, you can draw just an arc rather than the full ellipse - the angles w1 and w2 are given in whole degree steps as per Figure 1, with any arc being drawn in an anti-clockwise direction.

The width of the line drawn depends on the setting of the **DefLine** command, while the way a line or box is drawn on the background depends on the setting of the **DrawMode** and **BkColor** properties.

The **PEllipse** command acts very much the same, except that the ellipses drawn are filled with a pattern defined using **Deffill**.

Example

```
OpenW 1
Ellipse 100, 100, 40, 20, 90, 180  // Draws a
 quarter arc...
DefLine 0, 10
Ellipse 100, 100, 80, 40           // ...inside a
 full ellipse.
DefLine 0, 1 : DefFill 5
PEllipse 100, 200, 80, 40          // Draws a
 filled ellipse...
DefLine 2 : DefFill 48
PEllipse 100, 200, 80, 40, 45, 90  // ...with a
 pie section.
```

Remarks

The current scaling depends of the form's **ScaleMode** setting.

The **Ellipse** and **PEllipse** commands use the old GDI library. For a smoother ellipse drawn using anti-aliasing, you can use Windows GDI+ library instead.

Known Issues

Note: When the radius *r* is declared as a **Byte** or **Short**/**Word** the ellipse isn't drawn; this can be got around by using **CSng**(*r*). **Double**, **Int32** and **Int64** variables are unaffected.

See Also

Circle, PCircle, ScaleMode

{Created by Sjouke Hamstra; Last updated: 17/12/2015 by James Gaite}

Pset Command

Purpose

Sets a graphic point.

Syntax

Pset x, y [, color]

Pset [**Step**] (x, y) [, color]

x,y:Single exp
color:iexp

Description

Pset *x, y, color* sets a graphic point at the coordinates x and y in color *color*. **Pset** can be used as an alternative to:

```
Color RGB(r, g, b) : Plot x, y
```

however, it will not change the current color.

Pset x, y or **Pset**(x, y) sets a point in the current foreground color.

Pset Step (dx, dy) sets a point in the current foreground color at a distance of dx, dy from the current position.

Pset Step (dx, dy), color sets a point in the *color* at a distance of dx, dy from the current position.

Example

```
OpenW # 1
Do
  Pset Rand(_X), Rand(_Y), Rand(_C) - 1
Until MouseK && 2
CloseW # 1
```

Fills the screen slowly with many multicolored points.

Remarks

In Windows the last point of a line isn't drawn. The following fixes this:

```
Line x0, y0, x1, y1 : Pset(x1, y1)
```

See Also

Color, Plot, Draw, Line, SetDraw, Point, PTst

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Plot Command

Purpose

Draws a point on the screen.

Syntax

Plot x, y

x, y:floating-point expression

Description

Plot x, y draws a point with coordinates x, y on the screen. The coordinate system depends on the **ScaleMode** setting.

Example

```
OpenW # 1
Local mk%, mx%, my%
DefMouse 2
Do
  Mouse mx%, my%, mk%
  If mk% & 1
    Color Rand (_C) - 1
    Plot mx%, my%
  EndIf
Until mk% %& 2
CloseW # 1
```

An infinite loop which draws a point at the current mouse position after each mouse button click.

See Also

Draw, Line, PolyLine, Preset, Pset, QBDraw, SetDraw

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Draw Command

Purpose

Draws a point or a line between two points on the screen.

Syntax

Draw [**To**] [x, y]

Draw [x1, y1] [**To** x2, y2][**To** x3, y3]...

Draw exp

Draw(i)

SetDraw

x,y,x1,y1,x2,y2,i:floating-point expression

exp: a mixture of sexp and aexp, whereby the first expression must be a sexp. The individual expressions are separated by a comma, semi-colon o apostrophe.

Description

Draw x, y is equivalent to the **Plot** command, that is, a point with the coordinates x, y is drawn on the screen. **Draw To** x, y draws a line between the point with the coordinates x, y and the last set point. It is irrelevant whether this point was set with **Plot**, **Line** or **Draw**.

Draw x1, y1 **To** x2,y2 is equivalent to the **Line** command. However, additional coordinates can also be added. It is therefore possible to draw polygons in this manner.

Draw exp enables definition of commands similar to certain LOGO graphic commands (turtle graphics) or HPGL Hewlett-Packard standard plotter language commands. It is possible, in this way, to move an imaginary pencil across the screen, drawing as needed. The parameters for individual commands are floating point numbers which can also be specified using strings. The following commands are available:

FD n	moves the 'pencil' n pixels 'forward'.
BK n	moves the 'pencil' n pixels 'backwards'.
SX x SY y	scales the 'pencil movement' for FD or BK by the factor given in x or y. The scaling can be turned off with SX 0 or SY 0.
LT w	turns the 'pencil' left by the angle w (in degrees).
RT w	the same to the right

TT w	moves the 'pencil' to an absolute angle (in degrees). The assignment for w is as follows:
	w = 0: up or north
	w = 90: right or east
	w = 180: down or south
	w = 270: left or west

MA x, y	moves the 'pencil' to absolute coordinates x and y.
DA x, y	moves the 'pencil' to absolute coordinates x and y, and then draws a line in current color from the last set position to point (x, y).
MR x, y	like MA, except that it moves relative to last position.
DR	like MR, except that it moves relative to last

x, y	position.
CO n	defines color n as drawing color.
PU	lifts the 'pencil' up.
PD	lowers the 'pencil' down.

Draw(i) is a function which, depending on i, returns the following values:

i = 0	x coordinate (floating point number)
i = 1	y coordinate (floating point number)
i = 2	angle in degrees (floating point number)
i = 3	scaling on the x axis (floating point number)
i = 4	scaling on the y axis (floating point number)
i = 5	pen status (-1 for PD and 0 for PU)

SetDraw sets various values in the **Draw** exp command. For example, **SetDraw** x, y, w is equivalent to **Draw** "MA", x, y"TT",w command.

Example

```
Dim a%, i%
OpenW # 1
Draw 100, 100
// sets a point at 100,100
//
Draw To 10, 10
// draws a line from 100,00 to 10,10
```

```
//
Draw 10, 10 To 20, 20 To 30, 30
// draws a line line from 10,10 to 20,20 and from
  // 20,20 to 30,30
//
Draw "ma 160,200 tt0"
// starts at 160,200 with angle 0
//
Print AT(40, 1); "Press any key"
KeyGet a%
Cls
For i% = 3 To 10
  corner(i%, 90) //raws a polygon with i corners
Next i%
Print AT(1, 1); "Press any key"
KeyGet a%
Cls
For i% = 0 To 359
  SetDraw 320, 200, i%
  Draw "fd 45 rt 90 fd 45 rt 90 fd 45 rt 90 fd 45"
  Draw "bk 90 rt 90 bk 90 rt 90 bk 90 rt 90 bk 90"
  Draw "fd 45 rt 90 fd 45 rt 90 fd 45 rt 90 fd 45"
  Draw "bk 90 rt 90 bk 90 rt 90 bk 90 rt 90 bk 90"
Next
Print AT(1, 1); "Close the Window"
Do : Sleep : Until Me Is Nothing

Procedure corner(n%, r%)
  Local i%
  For i% = 1 To n%
    Draw "fd", r%, "rt", 360 / n%
  Next i%
Return
```

Draws a small and a large rectangle which both rotate around their own axis.

Remarks

ScaleMode determines the coordinate units.

See Also

Plot, Line, ScaleMode, QBDraw, Preset, Pset, SetDraw

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

SetDraw Command

Purpose

Sets the start position of the command **Draw.**

Syntax

SetDraw x, y, angle

x, y:Single exp
angle:iexp

Description

SetDraw x, y sets the initial position (x, y) and angle (degrees) to start drawing using **Draw**.

Example

Example 1:

```
OpenW 1
Local x%
SetDraw 100, 100, 0
'Draw "MA100,100,TT90"
// a little square
Draw"fd10rt90fd10rt90fd10rt90fd10rt90"
```

Example 2:

```
OpenW 1
Local x%, i%
SetDraw 100, 100, 0
For i% = 0 To 180
```

```
  SetDraw 320, 200, i%
  Draw "fd45rt90fd45rt90fd45rt90fd45"
  Draw "bk90rt90bk90rt90bk90rt90bk90"
  Draw "fd45rt90fd45rt90fd45rt90fd45"
  Draw "bk90rt90bk90rt90bk90rt90bk90"
  If i% = 180 Then i% = 0
  If i% = 0 Then Cls
  If i% = 0 Then x%++
  If x% > 80 Then Exit For
Next
```

Remarks

SetDraw 100, 100, 90 is a shortcut for **Draw** "MA100,100,TT90".

See Also

Draw, Line, Plot, PolyLine, Preset, Pset, QBDraw

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Point, RGBPoint and PTst Functions

Purpose

Returns the color of a point.

Syntax

rgb% = **Point**(x,y)

rgb% = **PTst**(x,y)

rgb% = **RGBPoint**(x,y)

x, y :floating point expression

Description

PTst(x, y), **RGBPoint**(x, y) and **Point**(x, y) are identical and all return the color of a point with the coordinates x, y, except the computer works from a palette (unlikely these days) where **Point** returns the palette number rather the colour itself.

Example

```
OpenW # 1
Local col%
Do
  If MouseK = 1
    Color Rand(_C) - 1
    Plot 100, 100
    Print AT(1, 1); Hex(Point(100, 100), 6);
      Space(2)//prints the color code of a set point
    col% = RGBPoint(100, 100) // or PTst(100,100)
      if you prefer
    Color 0
    Print AT(1, 2); "Red: "; Hex(GetRValue(col%),
      2); " "
    Print AT(1, 3); "Green: "; Hex(GetGValue(col%),
      2); " "
    Print AT(1, 4); "Blue: "; Hex(GetBValue(col%),
      2); " "
    While MouseK = 1 : Wend
  EndIf
  DoEvents
Until MouseK = 2 Or Win_1 Is Nothing
CloseW # 1
```

Remarks

See Also

-

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

QBDraw Command

Purpose

Draws a line or point with current graphics settings.

Syntax

QBDraw sexp

Description

QBDraw is a Quick Basic compatible Draw command.

Command strings are:

Un - Up, draws a line up around n units

Ln - Left, draws a line around n units to the left

Rn - Right, draws a line around n units to the rigth

Dn - Down, draws a line down around n units

En - draws a line around n units to the right above

Fn - draws a line around n units to the right below

Gn - draws a line around n units to the left below

Hn - draws a line around n units to the left above

Mi,,j - draw a line to i, j

M+n, m - with sign a relative line is drawn, M-n, m - e.g.: M-9,0 = U9

B Prefix - next command (ULRDEFGHM) doesn’t draw

N Prefix - next command (ULRDEFGHM) draws, but doesn‘t change the saved position

An -Turn, A0 = normally, A1 = turn to the left, A2 = turn on the head, A3 = turn to the right.

TAn - Turn in angle, units are given in degree, (A2 = TA180). With TA you are being able to turn the graphics created with the QBDraw command in degree steps.

Sn - Scaling. The given step width in (n, m) are multiplied with the scaling factor and after this diveded by four.
S4 (or S) are represent the normal or default condition, S8 correspond to a size doubling, S2 one bisection.

QBDraw uses integer coordinates.

Example

```
Draw 100, 100 //Position set
// draw a star with eight corners
// the 6 with e f g h is approximate Sin(Deg(45))
  * 8
QBDraw "nu8nl8nd8nr8ne6nf6ng6nh6"
Dim i As Integer, n As Integer, x As Double
Draw "ma100,100tt0"
n = 20
x = 360 / n
For i = 1 To n
  Draw "fd9rt"; x
Next i
```

See Also

Draw, SetDraw, Plot

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Curve Command

Purpose

draws a Bezier curve.

Syntax

Curve x0,y0,x1,y1,x2,y2,x3,y3

x0,y0,x1,y1,x2,y2,x3,y3: Single exp

Description

Curve x0,y0,x1,y1,x2,y2,x3,y3 draws a Bezier curve. The Bezier curve starts at x0,y0 and ends at x3,y3. At x0,y0 the curve is a tangent to the line from x0,y0 to x1,y1 and at x3,y3 a tangent to the line from x3,x3 to x2,y2.

If points x0,y0,...,x3,y3 are viewed as corners of a rectangle, the curve lies fully within this rectangle. (The curve can also be seen as a line between x0,y0 and x3,y3 which is pushed away from the points x1,y1 and x2,y2).

Example

```
OpenW # 1
Curve 10, 10, 10, 100, 100, 100, 100, 100
```

See Also

-

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

PolyLine, PolyFill Commands

Purpose

Draws connected lines with an arbitrary number of corners.

Syntax

PolyFill n, x(), y() [**OffSet** x0,y0]

PolyLine n, x(), y() [**OffSet** x0,y0]

niexp
x0, y0:floating-point expression
x(), y():avar floating-point array

Description

PolyLine n, x(),y() [OffSet x0,y0] draws connected lines with n corners. The x,y coordinates of the corner points are in arrays x() and y(). The first corner point is defined in x(0),y(0) and the last in x(n-1),y(n-1). The first and last corner points are automatically connected. Optionally, a horizontal and/or vertical offset (x0 or y0) can be added to these coordinates.

Polyfill works in the same way and fills the drawn polygon with the colour and/or pattern defined by **DefFill**.

Use caution when using **Option Base 1**; if an array has been defined to start at element one, then the first corner will be stored in x(1), y(1) rather than x(0), y(0) and the last corner in x(n), y(n).

Example

```
Option Base 0
OpenW # 1
Dim x!(3), y!(3), a%, i%
// Draws a triangle
Data 120,120,170,170,70,170,120,120
For i% = 0 To 3
  Read x(i%), y(i%)
Next i%
PolyLine 4, x(), y()
// Draw two filled stars, offset horizontally and
  vertically
Option Base 1
Data -59,-81,0,100,59,-81,-95,31,95,31
Dim x1!(5), y1!(5)
For i% = 1 To 5
  Read x1(i%), y1(i%)
Next i%
DefFill 5
QBColor 0
DefFill 10
PolyFill 5, x1(), y1() Offset ScaleWidth / 4,
  ScaleHeight * 2 / 3
DefFill 2
PolyFill 5, x1(), y1() Offset 3 * ScaleWidth / 4,
  ScaleHeight * 2 / 3
Do : Sleep : Until Me Is Nothing
```

See Also

Box, RBox, BkColor, DefFill, DefLine, DrawMode, RBox, PRBox, Box3D, PBox3D

{Created by Sjouke Hamstra; Last updated: 10/01/2016 by James Gaite}

OcxScale Property (Form Object)

Purpose

Sets or returns a value that determines the scaling units for Ocx controls.

Syntax

[Form.]**OcxScale** [= True | False]

Description

When **OcxScale** = True the coordinates of the Ocx controls are expected to be in the current **ScaleMode**. When **OcxScale** = 0 (False) the Ocx coordinates are expected in pixels (default).

Example

```
OpenW 1
Ocx Command cmd0 = "Normal", 10, 10, 80, 24
ScaleMode = basTwips
OcxScale = True
Ocx Command cmd1 = "Very Small", 10, 10, 180, 124
Do
  Sleep
Until Me Is Nothing
```

See Also

Form

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

ScaleX, ScaleY Functions

Purpose

Converts the value for the width or height of a **Form** or **Printer** from one of the **ScaleMode** property's unit of measure to another. Named arguments are not supported.

Syntax

x! = [*object*.]**ScaleX** *(width* [*, from*] [*, to*]*)*

y! = [*object*.]**ScaleY** *(height,* [*, from*] [*, to*]*)*

x!, y!	*: Single exp*
width, height	*: Single exp*
from, to	*: iexp, ScaleMode constant*

Description

The **ScaleX** and **ScaleY** methods take a value (*width* or *height*), with its unit of measure specified by *from*, and convert it to the corresponding value for the unit of measure specified by *to*.

The *height* and *width* parameters specify the number of units to be converted. The optional parameter *from* is a constant or value specifying the coordinate system from which width or height of object is to be converted. The optional parameter *to* is a constant or value specifying the coordinate system to which width or height of object is to be converted.

The possible values of *from* and *to* are the same as for the **ScaleMode** property: **basUser**, **basTwips**, **basPoints**, **basPixels**, **basCharacters**, **basInches**, **basMillimeters**, **basCentimeters**, and **basHiMetric**.

When *from* or *to* is omitted then the defaults are: *from* = **basHiMetric** and *to* = **basUser**. If one of the parameters **basUser**, then the value is converted using the current active **Scale** or **ScaleMode** setting of the Form or Printer.

Example

```
Local a$, i As Int, j As Int
AutoRedraw = 1
QBColor 0, 15
Cls
Restore
FontBold = True : Print "FROM \ TO"; : FontBold =
  False
For i = 1 To 8 : Read a$ : Print _Tab(i * 13); :
  FontBold = True : Print a$; : FontBold = False :
  Next i : Print
Restore
For i = 1 To 8 : Read a$
  FontBold = True : Print a$; : FontBold = False
  For j = 1 To 8
    QBColor i = j ? 13 : 0
    If i = 3 Or j = 3 : QBColor , 7
      a$ = Space(25) : Lset a$ = ScaleX(1, i, j)
    Else : QBColor , 15 : a$ = ScaleX(1, i, j)
    EndIf
    Print _Tab(j * 13); a$;
    QBColor , 15
  Next
  Print
Next
```

```
Data
  "Twips","Points","Pixels","Characters","Inches","
  Millimeters","Centimeters","HiMetric"
Do
  Sleep
Until Me Is Nothing
```

See Also

Form, Printer, ScaleMode, ScaleMode$, Scale, ScaleWidth, ScaleHeight, OcxScale

{Created by Sjouke Hamstra; Last updated: 17/05/2017 by James Gaite}

RubberBox Command

Purpose

Cuts out a rectangular segment of the screen.

Syntax

RubberBox x, y, minw, minh, varw, varh

x, y, minw, minhSingle exp
varw, varh:Single variables

Description

RubberBox can only be used by pressing the left mouse button.

Given are the coordinates of the upper left corner as well as the minimal width and height. By moving the mouse the size of the rectangle can be changed (rubber band effect) as long as the left mouse button is held down. When the mouse button is released the width and height are returned.

By specifying the negative width and height the rectangle can be drawn in the upper left direction.

Example

```
OpenW 1
Local Single a, b, x, y
Local k%
DefFill 4
Scale 0, 0, .5, .5
```

```
Do
  DoEvents
  Repeat
    Mouse x, y, k%
  Until k%
  Exit Do If k% = 2
  RubberBox x, y, 0, 0, a, b
  Color Rand(_C), Rand(_C)
  PBox x, y, x + a, y + b
Loop
CloseW # 1
```

This program enables drawing of rectangles in different colors with the mouse.

See Also

DragBox

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

DragBox Command

Purpose

Moves a sizing rectangle around the screen.

Syntax

DragBox x1,y1,w1,h1 [,x2,y2,w2,h2], x3,y3

x1,y1,w1,h1,x2,y2,w2,h2,x3,y3: single exp

Description

DragBox creates a rectangular cut-out with the width w1 and height h1, whose upper left corner is specified with x1 and y1. This rectangle can be moved within another rectangle by holding down the left mouse button and moving the mouse. The upper left corner of the second rectangle is given in x2 and y2, the width in w2 and the height in h2. When the movement is finished, x3 and y3 contain the coordinates of the upper left corner of moved rectangle.

Example

```
OpenW Full 1 : Win_1.AutoRedraw = 1
Global Single x1, y1, x2, y2, w1, h1, w2, h2, x3,
  y3
x1 = 20
y1 = 20
w1 = 100
h1 = 100
x2 = 10
```

```
y2 = 10
w2 = _X // horizontal width in pixels
h2 = _Y // vertical height in pixels
Do
  If MouseK And 1
    DragBox 20, 20, 100, 100, x3, y3
    // same as
    // DragBox 20, 20, 100, 100, x2, y2, w2, h2,
      x3, y3
    x1 = x3
    y1 = y3
    Box x1, y1, Add(x1, w1), Add(y1, h1)
    Print "ok"
  EndIf
Until MouseK And 2
CloseW 1
```

See Also

RubberBox

{Created by Sjouke Hamstra; Last updated: 28/11/2015 by James Gaite}

rc_InterSect Function

Purpose

Determines the overlapping area between two rectangles.

Syntax

fl! = **rc_InterSect**(x1,y1,w1,h1,x2,y2,w2,h2)

fl!:Boolean variable
x1,y1,w1,h1:integer expression;
x2,y2,w2,h2:variable names; return values

Description

The **rc_InterSect**() function tests if two rectangles overlap. The upper left corner of the first rectangle is specified in x1 and y1, the width in w1 and the height in h1.

The upper left corner of the second rectangle is specified in x2 and y2, the width in w2 and the height in h2. If the two rectangles overlap the function returns True (-1), otherwise it returns a False (0).

The upper left corner of the overlapping area between the two rectangles is returned in x2 and y2, the width in w2 and the height in h2. Because of this the last four parameters in the **rc_InterSect** function must always be integer variables (ByRef parameter).

If the two rectangles do not overlap, the x2, y2, w2, and h2 variables contain the coordinates of a rectangle between the two given rectangles. The width and height are then either negative or 0.

The first four parameters can also be specified with expressions. The last four parameters must be given as variables. They are changed by **rc_InterSect**.

Example

```
Auto a%, h%, w%, x%, y%
OpenW # 1
Box 10, 10, 400, 200
x% = 50 : y% = 50 : w% = 400 : h% = 400
Box x%, y%, x% + w%, y% + h%
If rc_InterSect(10, 10, 400, 200, x%, _
  y%, w%, h%)
  DefFill 4
  PBox x%, y%, x% + w% - 10, y% + h% - 10
EndIf
```

Draws two rectangles and fills the overlappingarea with a pattern.

See Also

-

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Clip Command

Purpose

Sets the bounds for graphic output.

Syntax

Clip x, y, w, h

Clip x1,y1 **To** x2,y2

Clip Off

x, y, w, h, x1,y1, x2,y2: floating point expression

Description

Clip x, y, w, h limits the graphic output to a defined rectangle.

Clip x1,y1 **To** x2,y2 defines the upper left corner of the clipping rectangle with x1 and y1, and the lower right corner with x2 and y2.

Clip Off turns the clipping off.

Clipping applies to the **AutoRedraw** bitmap as well. Clipping affects output in the Paint event, so that in case of an AutoRedraw the memory device context bitmap is copied for the clipping only. Make sure the clipping is off in this case.

Example

```
OpenW # 1 : Win_1.AutoRedraw = 1
PCircle 80, 80, 70  // Draws a full black circle
Clip 10, 10, 70, 70
// limits the graphic output to a window with the
  following coordinates:
// 10,10 upper left
// 80,10 upper right
// 10,80 lower left
// 80,80 lower right
Color 255
PCircle 80, 80, 70  // Draws a red top left
  quadrant
Clip Off // turns the clipping off.
```

Remarks

The clipping does not apply to the **Get** and **Put** commands.

In contrast to GFA-BASIC for Windows (16 Bit) version the **OffSet** of the **Clip** function is gone. In the GFA-BASIC 32 the offset in set using **ScaleLeft** and **ScaleTop**.

See Also

Scale, ScaleLeft, ScaleTop

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Get Command

Purpose

saves a portion of the screen in a string variable or a GDI bitmap.

Syntax

Get x1,y1,x2,y2, screensegment$

Get x1,y1,x2,y2, hbitmap

screensegment$:svar
hbitmap:Handle, ivar

Description

Get x1,y1,x2,y2, screensegment$ copies a portion of the screen with coordinates x1,y1 (upper left corner) and x2,y2 (lower right corner) to the string variable *screensegment$*.

Get x1,y1,x2,y2, hbitmap creates a device dependent bitmap with handle *hbitmap*.

Example

```
FullW 1
AutoRedraw = True
Local a$, a%, s%
BackColor = RGB(0, 255, 255)
ForeColor = RGB(255, 0, 0)
DefFill 5
PBox 10, 10, 100, 100
```

```
Get 10, 10, 100, 100, a$
FontTransparent = True
FontSize = 30
Text 20, 40, "Get"
Text 40, _Y - 300, "Please press key 'w'"
KeyGet s%
For a% = 1 To 700 Step 100
  Put 100 + a%, 100, a$
  Text 120 + a%, 140, "Put"
  Text 40, _Y - 300, "Please press key 'w'"
  KeyGet s%
Next
Cls
FontSize = 50 : FontBold = True
Text 20, _Y - 200, "End with Alt + F4"
Do : Sleep : Until Me Is Nothing
CloseW # 1
```

Draws a filled rectangle and copies a portion of this rectangle to a$. Using Put the rectangle is then returned to the window.

Remarks

The screen segments obtained with **Get** can be copied back to the screen by using **Put**.

When **Get** ,,,, hbitmap is used the bitmap must be released with **FreeBmp**; however, if that handle has been used to create a picture object by using **CreatePicture**, then the handle should not be freed until after you have finished with the picture object; otherwise, the handle which forms the source of the picture object will be destroyed and, thus, the picture will no longer be displayed.

GFA-BASIC 32 also supports the conversion of normal API bitmaps to an OLE Picture object with the **CreatePicture** function.

See Also

Put, FreeBmp, CreatePicture

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Put Command

Purpose

Copies a bitmap to the current output device.

Syntax

Put x, y, screensegment$[,mode]

Put x, y, hbitmap%[,mode]

x, y:Single expression
screensegment$:svar
hbitmap:Handle
mode:iexp

Description

Put x, y copies a portion of the screen saved with, for instance, **Get** back to screen memory, so that the upper left corner of the segment is aligned with the x,y coordinates on the screen.

By specifying the optional parameter mode it can be determined how the raster operation is to be performed. Raster operation codes define how the system combines colors in output operations that involve a brush, a source bitmap, and a destination bitmap. See BitBlt for a list of common raster operations.

Example

```
Auto a$, mk%, mx%, my%
```

```
OpenW # 1
DefFill 2
PBox 10, 10, 20, 20
Get 10, 10, 20, 20, a$
Repeat
  Mouse mx%, my%, mk%
  If mk% = 1
    Put mx%, my%, a$
  EndIf
Until mk% = 2
CloseW # 1
```

Draws a filled rectangle and saves it in the variables a$. When the left mouse button is pressed, the rectangle is moved to the current mouse position on the screen.

See Also

Get, Bitblt

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

BitBlt Command

Purpose

Copies a source raster (Bitmap) to a destination raster.

Syntax

BitBlt srcdc%,srcx%,srcy%,srcw%,srch%,
dstdc%,dstx%,dsty%,mode%

Description

BitBlt performs a fast copy of a source raster to a destination raster. To do this it requires the Device Context (srcdc%) of the source raster , the coordinates of the upper left corner (srcx%,srcy%), the width (srcw%), the height (srch%) and the Device Context (dstdc%) of the destination raster. The source raster is then moved to the location dstx%, dsty% whereby the source and destination raster as well as the current pattern specified in mode% can logically be combined. mode% must assume one of the following values:

BLACKNESS	$00000042	All bits are set to black
DSTINVERT	$00550009	The destination raster bits are inverted.
MERGECOPY	$00C000CA	The fill pattern is logically "And-ed" with the source raster.
MERGEPAINT	$00BB0226	The inverted source raster is logically "Or-ed" with the destination raster.

NOTSRCCOPY	$00330008	The inverted source raster is copied to the destination raster.
NOTSRCERASE	$001100A6	The source and destination raster are first "Or-ed". The resulting bit pattern is then inverted.
PATCOPY	$00F00021	The fill pattern is copied to the destination raster.
PATINVERT	$005A0049	The fill pattern is "Xor-ed" with the destination raster.
PATPAINT	$00FB0A09	The inverted source raster is first "Or-ed" with the fill pattern. The resulting bit pattern is then "Or-ed" with the destination raster.
SCRAND	$008800C6	The source and destination raster are "And-ed".
SRCCOPY	$00CC0020	The source raster is copied to the destination raster.
SRCERASE	$00440328	The source raster is "And-ed" with the inverted destination raster.
SRCINVERT	$00660046	The source and destination raster are "Xor-ed".
SRCPAINT	$00EE0086	The source and destination raster are "Or-ed".
WHITENESS	$00FF0062	All "white" bits are set.

Example

```
OpenW 3, 600, 0, 300, 300
OpenW 2, 300, 0, 300, 300
OpenW 1, 0, 0, 300, 300
AutoRedraw = True
Local pic As Picture, x%, b%, bmp%
b% = 300
For x% = 0 To 500
  Color Rand(_C)
  Line Rand(b%), Rand(b%), Rand(b%), Rand(b%)
Next
BitBlt Win_1.hDC, 0, 0, 300, 300, Win_2.hDC, 0, 0,
  SRCCOPY
// An alternative method...
Get 0, 0, 300, 300, bmp%
Set pic = CreatePicture(bmp%, 1)
Win_3.Picture = pic
Do : Sleep : Until MouseK = 2 /* Right-click to
  close windows
CloseW 1
CloseW 2
CloseW 3
```

Remarks

BitBlt corresponds to Windows function *BitBlt*().

Warning: **BitBlt** gets the Source-DC first and then the Dest-DC. This order is different from the order in operating system calls *BitBlt*(), *StretchBlt*(), and *PatBlt*().

If you got problems with **BitBlt** on a PC under Windows 98, you can solve it with an empty loop to insert a small delay. This problem comes from the driver of your graphic adapter.

See Also

FreeBmp, Patblt, Stretch

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

PatBlt Command

Purpose

Combines the given rectangle with the current fill pattern.

Syntax

PatBlt x, y, w, h, mode%

x, y, w, h:floating-point exp
mode%:integer expression

Description

PatBlt *x, y, w, h* combines the given rectangle with the current fill pattern. *mode* specifies the type of operation and must take one of the following values:

BLACKNESS	($00000042)	all "black" bits are set.
DSTINVERT	($00550009)	the destination raster bits are inverted.
PATCOPY	($00F00021)	the fill pattern is copied to the destination raster.
PATINVERT	($005A0049)	the fill pattern is "Xor-ed" with the destination raster.
WHITENESS	($00FF0062)	all "white" bits are set.

Example

```
OpenW # 1
Local a%
DefFill 30
```

```
// a canvas of 50x50
Win_1.ScaleWidth = 50
Win_1.ScaleHeight = 50
// PatBlt uses Me (or Output)
PatBlt 1, 2, 16, 16, PATCOPY
```

PatBlt is used here to copy the current fill pattern (defined with **DefFill**) to the screen. The upper left corner is located at (1,2) and the right corner at 17, 18. The PATCOPY mode copies the pattern without any logical operations (And, Or, Xor ...).

Remarks

PatBlt corresponds to Windows function PatBlt().

See Also

BitBlt, Stretch, DefFill

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Stretch Command

Purpose

Copies a bitmap into a destination rectangle, stretching or compressing the bitmap to fit the dimensions of the destination rectangle, if necessary.

Syntax

Stretch x, y, a$, w, h [, Mod]

Stretch x, y, hBmp, w, h [, Mod]

x, y, w, h, Mod:integer expression
a$:svar; bitmap
hBmp:integer expression; bitmap handle

Description

Stretch copies the bitmap specified in a$ to the coordinates specified in *x* and *y*. The bitmap must first be read into the variable a$ by using **Get**.

hBmp can be used instead of *a$*. In this case a handle, obtained from **Get** or **LoadImage**, or a handle from a **Picture** object, is passed.

w and *h* specify the width and height of the destination area. If the dimensions of the source area are greater than the destination area the bitmap is correspondingly shrunk. In the reverse case, i.e. the dimensions of the source area are smaller than that of the destination area the bitmap is correspondingly stretched.

During the copy the source raster, the destination raster and the current fill pattern can be combined with each other. *Mod* must then take of the following values:

BLACKNESS($00000042)

DSTINVERT($00550009)

MERGECOPY($00C000CA)

MERGEPAINT($00BB0226)

NOTSRCCOPY($00330008)

NOTSRCERASE($001100A6)

PATCOPY($00F00021)

PATINVERT($005A0049)

PATPAINT($00FB0A09)

SCRAND($008800C6)

SRCCOPY($00CC0020)

SRCERASE($00440328)

SRCINVERT($00660046)

SRCPAINT($00EE0086)

WHITENESS($00FF0062)

For Description of the values see command **BitBlt**

Example

```
Local a$, a%, n%
Local x%(5), y%(5)
For n% = 0 To 4
  Read x%(n%), y%(n%)
Next n%
OpenW 1
PolyFill 5, x%(), y%() Offset _X / 2, _Y / 2
Message "Click OK to continue"
Get _X / 2 - 96, _Y / 2 - 82, _X / 2 + 96, _Y / 2
  + 100, a$
Stretch 0, 0, a$, 96, 91
Data -59,-81,0,100,59,-81,-95,31,95,31
```

Draws a star, gets it in a string (a$) and puts it back using **Stretch** at position (0,0), using half of the width and half of the size. The picture is really sized to size 96x91 instead of the original 192x182.

```
AutoRedraw = 1
Ocx CommDlg cd
With cd
  .Filter = "*.bmp;*.gif;*.jpg"
  .FileName = "*.bmp;*.gif;*.jpg"
  .IniDir = WinDir
  .ShowOpen
EndWith
Dim pic As Picture
If Exist(cd.FileName)
  Set pic = LoadPicture(cd.FileName)
  Stretch 0, 0, pic.Handle, _X, _Y
  Set pic = Nothing
EndIf
```

Remarks

Stretch corresponds to Windows function *StretchBlt*().

See Also

CreatePicture, LoadPicture, PaintPicture, FreeBmp, BitBlt, PatBlt

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

FreeBmp Command

Purpose

Deletes from memory a bitmap created with **Get** or the API function *CreateDIBSection*.

Syntax

FreeBmp hBmp

hBmp:Handle, integer expression

Description

With **Get** or the Windows API function *CreateDIBSection* you can create a bitmap and retrieve its handle. This is a good way to place many pictures in memory. However, once a bitmap is no longer needed, you must free its handle with **FreeBmp**. Note: Each use of the **Get** command which returns a handle requires a matching **FreeBmp** statement to free the memory, otherwise you will get a Memory leak.

However, if the handle produced with **Get** is then used, through **CreatePicture**, to create a picture object, **FreeBmp** should not be used until after you have finished with the picture object; otherwise, it will delete the handle which forms the source of the picture and, thus, the picture will not be shown.

Example

```
OpenW 1
Global pict%, i%
```

```
Ocx Command cmd1 = "Exit", 10, 10, 100, 40
Win_1.AutoClose = False
For i% = 1 To 30
  Circle 200, 200, 10 + i% * 2
  Color i% * 1000
Next
Get 120, 120, 280, 280, pict%
Line 0, 119, _X, 119
Line 0, 281, _X, 281
Put 300, 120, pict%
Do
  Sleep
Until Me Is Nothing

Sub cmd1_Click
  PostMessage Win_1.hWnd, WM_CLOSE, 0, 0
EndSub

Sub Win_1_Close(Cancel?)
  If Message("Really Quit??", , MB_YESNO) = IDYES
    Cancel? = False
    FreeBmp(pict%)
  EndIf
EndSub
```

See Also

Get, Put

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Font, StdFont Object

Purpose

The **Font** object contains information needed to format text for display in the interface of an application or for printed output.

Syntax

Dim *name* **As Font**

Dim *name* **As** [**New**] **StdFont**

Description

You frequently identify a **Font** object using the **Font** property of an object that displays text (such as a **Form** object or the **Printer** object).

You cannot create a **Font** object using code like Dim X As New Font. If you want to create a **Font** object, you must use the **StdFont** object like this:

Dim X As New StdFont

Properties

Bold	Bool	get/put	Returns or sets the font style to either bold or non bold.
CharSet	Short	get/put	Sets or returns the character set used in the font.

			0 - Standard Windows characters 2 - The symbol character set. 128 - Double-byte character set (DBCS) unique to the Japanese version of Windows 255 - Extended characters normally displayed by DOS applications.
Italic	Bool	get/put	Returns or sets the font style to either italic or non-italic.
Name	String	get/put	Returns or sets the name of a font.
Size	Currency	get/put	Returns or sets the font size used in points.
Strikethrough	Bool	get/put	Returns or sets the font style to either strikethrough or non-strikethrough
Underline	Bool	get/put	Returns or sets the font style to either underlined or non-underlined
Weight	Short	get/put	Returns or sets the weight of the characters. The weight refers to the thickness of the characters, or the “boldness factor”. The higher the value,

			the bolder the character.
_hFont	Handle	Get	Returns the font handle.

Example

If you put a **TextBox** control named Text1 on a form, you can dynamically change its **Font** object to another using the **Set** statement, as in the following example:

```
Ocx TextBox Text1 = "Hello", 10, 10, 150, 35 :
  Text1.BorderStyle = 1
Dim X As New StdFont
X.Bold = True
X.Name = "Arial"
X.Size = 16
X.Strikethrough = True
Set Text1.Font = X
Do : Sleep : Until Me Is Nothing
```

Remarks

As an alternative, the following can be used:

```
Text1.FontBold = True
Text1.FontStrikeThrough = True
Text1.FontSize = 16
```

More information about fonts can be gleaned through using the **GetTextMetrics**() API as shown below:

```
Type TEXTMETRIC
  tmHeight As Long
  tmAscent As Long
  tmDescent As Long
  tmInternalLeading As Long
  tmExternalLeading As Long
  tmAveCharWidth As Long
  tmMaxCharWidth As Long
  tmWeight As Long
  tmOverhang As Long
  tmDigitizedAspectX As Long
  tmDigitizedAspectY As Long
  tmFirstChar As Byte
  tmLastChar As Byte
  tmDefaultChar As Byte
  tmBreakChar As Byte
  tmItalic As Byte
  tmUnderlined As Byte
  tmStruckOut As Byte
  tmPitchAndFamily As Byte
  tmCharSet As Byte
End Type
Local tm As TEXTMETRIC
OpenW 1 : Win_1.FontName = "Courier New"
~GetTextMetrics(Win_1.hDC, tm)
Print "TextHeight: "; tm.tmHeight
Print "Font Ascent (above baseline):"; tm.tmAscent
Print "Font Descent (below baseline):";
  tm.tmDescent
```

See Also

Font Property, Setfont, Freefont, RFont, _Font$

{Created by Sjouke Hamstra; Last updated: 14/01/2015 by James Gaite}

Font To Command

Purpose

Generates font parameters for **SetFont**

Syntax

Font *keyword value* [**To** hFont]

keyword:font attribute name
value:attribute setting
hFont:Handle

Description

By using the **Font** command a font other than the standard Windows font can be generated for **SetFont** (e.g. SYSTEM_FIXED_FONT). The parameters are quite numerous and the syntax is fairly flexible. **Font** is followed, in addition to many programming lines, by a number of keywords which are themselves followed by a parameter:

ITALIC n	n = 0 normal font n <> 0 italic font
UNDERLINE n	n = 0 normal font n <> 0 underlined
STRIKEOUT n	n = 0 normal font n <> 0 strikeout

CHARSET n	n = 0 ANSI_CHARSET - Windows char set
	n = 2 SYMBOL_CHARSET - symbol character set (Greek, mathematical or

	dingbats)
	n = 128 SHIFTJIS_CHARSET - Japanese
	n = 255 OEM_CHARSET - IBM character set

OUTPRECISION n	At this time not implemented in Windows
	n = 0 OUT_DEFAULT_PRECIS
	n = 1 OUT_STRING_PRECIS
	n = 2 OUT_CHARACTER_PRECIS
	n = 3 OUT_STROKE_PRECIS

CLIPPRECISION n	regulates the clipping of characters which are partially outside of the Clipping area.
	n = 0 CLIP_DEFAULT_PRECIS
	n = 1 OUT_CHARACTER_PRECIS
	n = 2 OUT_STROKE_PRECIS

QUALITY n	determines whether the Windows bitmaps are scaledin order to generate other font sizes.
	n = 0 DEFAULT_QUALITY
	n = 1 DRAFT_QUALITY
	n = 2 PROOF_QUALITY (letter quality,

PITCH n	specifies proportional (i.e., the "i" occupies asmaller character width than the "m".)
	n = 0 DEFAULT_PITCH
	n = 1 FIXED_PITCH
	n = 2 VARIABLE_PITCH

FAMILY n	font family: n = 0 FF_DONTCARE doesn't matter, like SYSTEM_FIXED_FONT n = 16 FF_ROMAN a font with serifs, small hooks n = 32 FF_SWISS a simple font without decorations

	n = 48 FF_MORN COURIER, PICA etc., similar to a typewriter font, fixed pitch, or OEM_FIXED_FONT
	n = 64 FF_SCRIPT longhand font
	n = 80 FF_DECORATIVE symbols, dingbats, Greek

WEIGHT n	light, normal or bold
	n = 0 FW_DONTCARE
	n = 100 FW_THIN
	n = 200 FW_EXTRALIGHT
	n = 300 FW_LIGHT
	n = 400 FW_NORMAL normal
	n = 500 FW_MEDIUM
	n = 600 FW_SEMIBOLD
	n = 700 FW_BOLDBOLD
	n = 800 FW_EXTRABOLD
	n = 900 FW_HEAVY

WIDTH n	character width
HEIGHT n	character height
ORIENTATION n	rotation angle of individual characters in 10ths of a degree, so for n = 1800 they are upside down
ESCAPEMENT n	character rotation again in 10ths of a degree, Orientation and Escapement are only available for vector fonts. "Morn", "Roman", and "Script".

Font To hFont**To** then follows as the last variable. It determines the creation of a logical font according the setting specified. This variable returns as a result a font handle. This font can then be used anywhere, where a font handle is required, for instance with **SetFont.** Afterwards, the font must be released with **FreeFont** or **DelFont**.

The parameters can span several lines (all starting with **Font**...) and may optionally be separated with commas.

Example

```
OpenW 1
Local fnt As Handle
Font "roman"
Font Italic 0, Weight 1000 , Width 20, Height 40
Font Orientation 0, StrikeOut 0, Underline 0,
  Escapement 450
Font Family FF_ROMAN, CharSet OEM_CHARSET , Pitch
  FIXED_PITCH
Font To fnt
SetFont fnt
Text 100, 100, "Hello"
SetFont SYSTEM_FONT
FreeFont fnt
```

Remarks

_Font$ returns a string with the parameters set with the **Font** command.

See Also

Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 06/10/2014 by James Gaite}

SetFont Method/Command

Purpose

Changes the font in the current **Form** or **Printer.**

Syntax

SetFont hFont (Command)

[*Object*].**SetFont** Name, Size, Bold, Italic, Underline, StrikeThru, CharSet (Method)

hFont:Handle
Object:Ocx Object
CharSet:Integer
Size:Single
Name:String
Bold, Italic, Underline, StrikeThru: Bool

Description

SetFont *hFont* selects a font using a font handle or a system constant. A font handle can be obtained using **Font To**, **Dlg Font**, **_font$**, or *CreateFontIndirect*(). **SetFont** *hFont* is a 16-bit compatible command.

ConstantDESCRIPTION

SYSTEM_FONT(13) - standard proportional font

SYSTEM_FIXED_FONT(16) - a similar non-proportional font

ANSI_VAR_FONT(12) - a Helvetica or Times font, SYSTEM_FONT, but a little smaller.

ANSI_FIXED_FONT(11) - a typewriter font (like Courier), a little smaller than SYSTEM_FIXED_FONT

DEVICE_DEFAULT_FONT(14) - can be any font, mostly SYSTEM_FONT, selected by the driver.

OEM_FIXED_FONT(10) - DOS window character set. However, instead of ANSI (WINDOWS) character set the OEM (read IBM) character set is used.

The second variant [Object.]**SetFont** is an Ocx method and supports a compact way of changing the current **Font** object. In contrast with the first variant, **SetFont** *hFont*, the **SetFont** method manipulates the current **Font** object of an Ocx object, or the current active **Form** or **Printer**.

Example

```
' AutoRedraw also opens the window Me
AutoRedraw = 1
Dim fnt As Handle, i As Int, s$
' Select a screen font
Dlg Font Me, 0, 1
' Create a handle
Font To fnt
' Read the name of a font
RFont Name s$
' Activate the font
SetFont fnt
' Test it
Print "test", s$
' Activate the SYSTEM_FONT
SetFont 13
' another test
Print "test"
' Give the used memory free
DelFont fnt
```

Remarks

The font handling for **Form** and **Printer** objects is very different from each other. API font handles (Font To, Dlg Font) should not be mixed with **Font** objects.

In contrast with GFA-BASIC 16 **SetFont** 0 is not allowed (crash).

See Also

Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

GetFont

Purpose

Reads the parameters for the given font.

Syntax

GetFont hFont

hFont: Handle

Description

This function reads the parameters for the font with the given font handle similar to the **Font...To** under the **Font** Command.

Example

```
Debug.Show
Debug
// prints only the font number
Trace SYSTEM_FONT
Debug
// prints all information, if different
Trace _Font$
GetFont SYSTEM_FONT
Trace _Font$
GetFont SYSTEM_FIXED_FONT
Trace _Font$
GetFont DEVICE_DEFAULT_FONT
Trace _Font$
GetFont OEM_FIXED_FONT
```

```
Trace _Font$
GetFont ANSI_FIXED_FONT
Trace _Font$
```

See Also

Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

RFont Command

Purpose

Reads the current font parameters returned by GetFont Command.

Syntax

RFont name var [, name var, ...]

name:font attribute name
var:variable

Description

This command is the opposite of the **Font** Command. It allows the current font parameters or the ones returned by **GetFont** to be read. The following alternatives are allowed:

RFont CharSet c|

RFont ClipPrecision c|

RFont Escapement c%

RFont Family c|

RFont Height c%

RFont Italic c|

RFont Name c$

RFont Orientation c%

RFont OutPrecision c|

RFont Pitch c|

RFont Quality c|

RFont StrikeOut c|

RFont Underline c|

RFont Weight c%

RFont Width c%

c|:Byte variable
c%:Integer variable
c$:String variable

For example, **RFont** Italic a| returns the Italic value for the font. You can use the same parameters as for the **Font** command, of course, followed by a variable after the keyword e.g. ITALIC. Instead of specifying a string variable for the name you can also use addr%. **Char{**addr%**}** will then read the name.

The preferred way to handle fonts in Forms and Ocx controls is by using the **Font** property of the objects.

Example

```
OpenW 1
Local a%, fnt_b%, fnt_i%, fnt_s%
Local fnt_u%, h&, org%, p$, p%, w&
org% = OEM_FIXED_FONT
'org% = SYSTEM_FONT
SetFont org%
GetFont org%
```

```
RFont Name p$ //Font Name
Font Italic 1
Font To fnt_i%
Font Underline 1, Italic 0
Font To fnt_u%
Font Underline 0
RFont Height h&, Width w&
Font Height h& / 4, Width w& / 4
Font To fnt_s%
Font Height h& * 7, Width w& * 7
Font To fnt_b%
Print p$; " font"; //Print example
SetFont fnt_i%
Print p$; " font italic ";
SetFont fnt_u%
Print p$; " font underline";
SetFont fnt_s%
Print p$; " font small"
Print
Print
SetFont fnt_b%
Print p$; " font big";
SetFont SYSTEM_FONT
FreeFont fnt_i%
FreeFont fnt_u%
FreeFont fnt_s%
FreeFont fnt_b%
```

Remarks

Internally, GFA-BASIC 32 maintains a LOGFONT structure which is filled using the **Font** command and read with **RFont**. The internal LOGFONT contains the values for the font of the current Form. The members of the LOGFONT reflect the settings of the Font object of the Form. After changing a Font property (**FontItalic** = True), the LOGFONT is updated to reflect the new settings.

The current LOGFONT settings can be saved using *logfont*$ = **_font$** and later reselected using **_font$** = *logfont*$. The Font object of the Form is then updated with the saved font settings.

See Also

Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Dlg Font Command

Purpose

Invokes the common font selecting dialog box.

Syntax

Dlg Font form, hDC, Flags[,Color[,Min, Max[,"Style"]]]

form:Form object
Dc, Flags, Color, Min, Max:integer expression

Description

This command calls the common font-selector Dialog in COMMDLG.DLL.

form is a form object, like **Me**, **Win_1**, **Dlg_1**, frm1.

Dc is a device context of a printer, will be used only when declared with corresponding flags. When using screen fonts only, Dc should beset to a Null.

Flags is a long integer, which gives the bit-wise parameter for the font selection. These bits are:

CF_SCREENFONTS	$000001	Indicates Screen font.
CF_PRINTERFONTS	$000002	Indicates Printer font.
CF_BOTH	$000003	Indicates Screen and Printer fonts.
CF_INITTOLOGFONTSTRUCT	$000040	Use the form's LOGFONT structure
CF_EFFECTS	$000100	Permits effects like: underlined, crossed out and color selections.
CF_APPLY	$000200	enables APPLY-button, with which the actual style and point size will be

		represented in the example field .
CF_ANSIONLY	$000400	Only enable fonts with the ANSI Characters set.
CF_NOVECTORFONTS	$000800	Only non-vectored fonts.
CF_NOSIMULATIONS	$001000	No GDI font simulations.
CF_LIMITSIZE	$002000	type-size limitation uses Min and Max parameters.
CF_FIXEDPITCHONLY	$004000	Only moonscape fonts.
CF_WYSIWYG	$008000	Only fonts that are available on the screen and the printer. Use with CF_BOTH and F_SCALABLEONLY.
CF_FORCEFONTEXIST	$010000	Show only fonts with a corresponding file. CF_SCALABLEONLY$020000 Only fonts which can assume any size (as vectored or TrueType)
CF_TTONLY	$040000	Only TrueType fonts (available in Windows 3.1 and higher)
CF_NOFACESEL	$080000	No Font selection. Used for selecting multiple fonts.
CF_NOSTYLESEL	$100000	No style selection. (i.e.: bold, italic...)
CF_NOSIZESEL	$200000	No size selection. This bit is automatically set, if "Style" is declared.
CF_USESTYLE	$000080	

These bits are not allowed in GFA-Basic:

CF_SHOWHELP	$000004
CF_ENABLEHOOK	$000008
CF_ENABLETEMPLATE	$000010
CF_ENABLETEMPLATEHANDLE	$000020

Color declares the color for the selected font. CF_EFFECTS must be set. Color value is must be stored in **_ECX.**

Min and *Max* are minimum and maximum point sizes. CF_LIMITSIZE must be set.

"Style" notes that the font style name is to be returned. If *"Style"* or *""* is declared, the pointer to the style name is placed in **_EBX**. (i.e. Print (Char**{**_EBX**}**) may display Bold Italic.) **_DX** holds the size of a font in tenths of a point. **_SI** holds the type of the selected fonts. The possible values can any combination of:

SIMULATED_FONTTYPE	$8000	GDI Simulated Font.
PRINTER_FONTTYPE	$4000	Printer Font.
SCREEN_FONTTYPE	$2000	Screen Font
BOLD_FONTTYPE	$0100	TrueType Bold Font.
ITALIC_FONTTYPE	$0200	TrueType Italic Font.
REGULAR_FONTTYPE	$0400	TrueType Regular Font.

In the font field, normal (Windows 3.0) Fonts will not be marked. TrueType Fonts will be marked with a double T and printer fonts with a small printer symbol.

_AX is a null if there is an error.

Example

```
OpenW 1
Global col%, fnt%
Dim a%(16), hfnt As Handle, s$
Dlg Font Win_1, 1, cdfScreenFonts, a%(1), col%
Font To hfnt          // create font handle
RFont Name s$         // obtain font name
SetFont hfnt          // select font
Print "test", s$      // test
SetFont 13            // select SystemFont
Print "test"          // test
DelFont fnt           // delete font object
```

Remarks

This command is implemented for compatibility reasons only. Use **CommDlg** object instead.

See Also

CommDlg, Dlg Color, Dlg Open, Dlg Print, Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

_hfont Function

Purpose

Returns the handle of the font currently selected in the active **Form** (OpenW, Dialog, Form)

Syntax

x = **_hfont**

x : Handle

Description

Using **_hfont** the handle of the current font of the current active window can be obtained. It could then be used to set the font of some custom, non-OCX, control using WM_SETFONT.

In case of a **Form**, **OpenW**, **ChildW**, and **ParentW** the **_hfont** returns the handle of a **StdFont** object. When this font object is destroyed the font handle is no longer valid.

Example

```
OpenW 1
AutoRedraw = 1
Print _hFont
Print Me.Font._hFont
Do
  Sleep
Until Me Is Nothing
```

Remarks

_hFont is a shortcut for **Me.Font._hFont**.

See Also

Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

_font$ Function

Purpose

Returns the font parameters of the internal font info set with **Font**, **GetFont**, **Dlg Font** and **_font$=** in a String.

Syntax

$ = **_font$**

Description

See description in _font$=

See Also

Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

_font$=

Purpose

Fills the internal font information with the contents of a string.

Syntax

**_font$=** a$

a$: svar

Description

Used to fill the internal LOGFONT structure with a previously saved string using **_font$**.

The contents of the _**font$** string could look like this:

SYSTEM_FONT	"System,16,w7,7,q2,f34,p513"
SYSTEM_FIXED_FONT	"Fixedsys,15,w8,4,q2,f49,p513"
ANSI_FIXED_FONT	"Courier,12,w9,q2,f1,p512"
ANSI_VAR_FONT	"MS Sans Serif,12,w9,q2,f2,p512"
OEM_FIXED_FONT	"Terminal,12,w8,c255,q2,f1,p512"
DEVICE_DEFAULT_FONT	",0,f1"

The string is build according the following format. First the name of the font followed by the character height (if necessary with sign). Then, optional and separated with commas the rest of the LOGFONT members. Zero values are left out, so ",i0" for ITALIC 0 is not included. Almost all values are prefixed with a character (i for ITALIC, w for WIDTH, etc), only WEIGHT and HEIGHT are not preceded with a character. Their position in the _font$ string determines their value.

Overview of the characters

w	WIDTH
e	ESCAPEMENT
o	ORIENTATION
	WEIGHT
i	ITALIC
u	UNDERLINE
s	STRIKEOUT
c	CHARSET
q	QUALITY
f	FAMILY+PITCH
p	PRECISION (OUTPRECISION + CLIPPRECISION*256)

Using **_font$=** all font parameters can be set in one instruction.

_font$="" clears all font parameters.

Example

```
OpenW 1
Print
GetFont SYSTEM_FONT
SetFont SYSTEM_FONT
Print "SYSTEM_Font", _Font$
GetFont SYSTEM_FIXED_FONT
SetFont SYSTEM_FIXED_FONT
Print "SYSTEM_FIXED_Font", _Font$
Do : Sleep : Until Me Is Nothing
```

```
CloseW 1
_Font$ = "Arial,48,7"   /* Bold Arial in 48 Pixel
 high
_Font$ = "Arial,48,700" /* Bold Arial in 48 Pixel
 high
_Font$ = ",48,7,f34"    /* Bold, Swiss-Family,
/* Variable-Pitch: Arial, Helvetica..., in 48 pixels
 high
_Font$ = ",-48,f49"     /* Morn fixed font
/* (usually Courier New), Text-Height 48 Pixel
_Font$ = ",0,c1"        /* some Font
_Font$ = ",0,c0"        /* some ANSI Font
_Font$ = ",0"           /* some ANSI font
_Font$ = ",0,c255"      /* some IBM-PC font
 (Terminal?)
```

The order of the parameters is not relevant, except for the font-name and the character height. White spaces are ignored.

After setting the font parameters the font can be selected into the current window using "**Font To** var%" and "**SetFont** var%"

Remarks

The Windows API function *GetTextFace*() may be used to determine the name of the actual used font:

```
_Font$ = "Arial,48,7"
SetFont _Font$
Local a$ = Space$(80)
~GetTextFace(_DC(), 80, V:a$)
Print a$
```

See Also

Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

FreeFont Command

Purpose

Releases a GDI font.

Syntax

FreeFont hFont

hFont:Handle

Description

Releases a font from memory. However, you MUST make sure that the font is not selected in a DC (Device Context) so be sure to perform a **SetFont** SYSTEM_FONT beforehand.

Example

```
OpenW 1
Global fnt1%, fnt2%, fnt3%, fnt4%, fnt5%, x%
fonts
Text 10, 10, "GFA Software Technologies 0"
SetFont fnt1%
Text 10, 30, "GFA Software Technologies 1"
SetFont fnt2%
Text 10, 80, "GFA Software Technologies 2"
SetFont fnt3%
Text 10, 120, "GFA Software Technologies 3"
SetFont fnt4%
Text 10, 160, "GFA Software Technologies 4"
SetFont fnt5%
```

```
Text 10, 200, "GFA Software Technologies 5"
SetFont SYSTEM_FONT
Text 10, 240, "GFA Software Technologies 6"
FreeFont fnt1%
FreeFont fnt2%
FreeFont fnt3%
FreeFont fnt4%
FreeFont fnt5%
Do : Sleep : Until Me Is Nothing

Procedure fonts
  Font Family 0, Quality 0
  Font "roman", Height 40, Width 0
  Font Weight FW_BOLD, Orientation 0
  Font Escapement 0, Italic 0, Underline 0
  Font StrikeOut 0, CharSet OEM_CHARSET
  Font To fnt1%
  Font Quality PROOF_QUALITY, Height 25
  Font CharSet ANSI_CHARSET, "Helv"
  Font To fnt2%
  Font CharSet OEM_CHARSET, "Script", Height 30
  Font To fnt3%
  Font "Morn", Height 50
  Font To fnt4%
  Font "symbol", Italic 0, Weight 1000, Width 25
  Font Height 40, Orientation 0
  Font StrikeOut 0, Underline 0, Escapement -150
  Font Family FF_ROMAN, CharSet OEM_CHARSET
  Font To fnt5%
EndProc
```

Remarks

DelFont is synonym to **FreeFont.**

See Also

Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

DelFont Command

Purpose

Deletes a logical font.

Syntax

DelFont hFont

hFont: Handle

Description

DelFont frees all system resources associated with the object. After the object is deleted, the specified handle is no longer valid. **DelFont** invokes *DeleteObject* API.

Example

```
Local fnt%
AutoRedraw = True
_Font$ = "Arial"
Font To fnt
SetFont fnt
Print "Hello World"
DelFont fnt
Do // to end press Alt + F4
  Sleep
Until Me Is Nothing
```

Remarks

DelFont is synonym to **FreeFont**.

See Also

Font, Font To, SetFont, GetFont, RFont, Dlg Font, _hFont, _font$, _font$=, FreeFont, DelFont

{Created by Sjouke Hamstra; Last updated: 30/09/2014 by James Gaite}

GdiFlush Command

Purpose

The **GdiFlush** function flushes the GDI graphical output that has been cached (batch).

Syntax

GdiFlush

Description

GDI batches drawing functions to enhances drawing performance by minimizing the amount of time needed to call GDI drawing functions that return **Boolean** values. The system accumulates the parameters for calls to these functions in the current batch and then calls the functions when the batch is flushed by any of the following means:

- Calling the GdiFlush function
- Reaching or exceeding the batch limit set by the *GdiSetBatchLimit* API function
- Filling the batching buffers.
- Calling any GDI function that does not return a Boolean value.

An application should call **GdiFlush** before a thread goes away if there is a possibility that there are pending function calls in the graphics batch queue. The system does not execute such batched functions when a thread goes away.

Remarks

See Also

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

HimetsToPixelX, HimetsToPixelY, PixelsToHimetX, PixelsToHimetY Function

Purpose

Converts between pixel and Himetric units.

Syntax

p# = **HimetsToPixelX**(h#)
p# = **HimetsToPixelY**(h#)

h# = **PixelsToHimetX**(p#)
h# = **PixelsToHimetY**(p#)

h, p : Double expression

Description

HimetsToPixelX and **PixelsToHimetX** convert on the horizontal plane.

HimetsToPixelY and **PixelsToHimetY** convert on the vertical plane.

A Himet (Himetric Unit - the internal OLE- base coordinates unit) is 1/100 mm. 1 Twips (the base unit of GFA-BASIC 32 OLE) is 1/20 Point = 1 /1440 inch. The conversion factor between Twips and Himets are constants: n Twips = n * 2540/1440 Himets.

Example

```
Debug.Show
Trace _X
Trace HimetsToPixelX(1)    // Pixels to Himets
  (horizontally)
Trace PixelsToHimetX(_X)   // Width in Himets
Trace _Y
Trace HimetsToPixelY(1)    // Pixels to Himets
  (vertically)
Trace PixelsToHimetY(_Y)   // Height in Himets
```

See Also

PixelsToTwipX(), PixelsToTwipY(), TwipsToPixelX(), TwipsToPixelY()

{Created by Sjouke Hamstra; Last updated: 03/03/2018 by James Gaite}

PixelsToTwipX, PixelsToTwipY, TwipsToPixelX, TwipsToPixelY Function

Purpose

Converts between pixels and twips.

Syntax

t# = **PixelsToTwipX**(p#)
t# = **PixelsToTwipY** (p#)

p# = **TwipsToPixelX**(t#)
p# = **TwipsToPixelY**(t#)

p, t : Double numeric expression

Description

With **PixelsToTwipX** and **TwipsToPixelX** values can be converted on the horizontal plane.

With **PixelsToTwipY** and **TwipsToPixelY** values can be converted on the vertical plane.

The Pixel to Twip and Twip to Pixel properties of the **Screen** property contain the conversion factors for these functions.

Example

```
Debug.Show
Trace _X
Trace TwipsToPixelX(1)    // Pixel to Twips
  (horizontally)
Trace PixelsToTwipX(_X)   // Width in Twips
Trace _Y
Trace TwipsToPixelY(1)    // Pixel to Twips
  (vertically)
Trace PixelsToTwipY(_Y)   // Height in Twips
```

See Also

HimetsToPixelX(), HimetsToPixelY(), PixelsToHimetX(), PixelsToHimetY(), Screen

{Created by Sjouke Hamstra; Last updated: 03/03/2018 by James Gaite}

Open Command

Purpose

Enables input/output (I/O) to a file or a peripheral device.

Syntax

Open *pathname* [**For** *mode*] [**Access** *access*] [share] [**Commit**] [**Based** 0/1] **As** [**#**]*filenumber* [**Len**=*reclength*]

Description

You must open a file before any I/O operation can be performed on it. **Open** allocates a buffer for I/O to the file and determines the mode of access to use with the buffer.

If the file specified by *pathname* doesn't exist, it is created when a file is opened for **Append**, **Binary**, **Output**, or **Random** modes.

If the file is already opened by another process and the specified type of access is not allowed, the **Open** operation fails and an error occurs.

pathname	Required. String expression that specifies a file name - may include directory or folder, and drive.
mode	Optional. Keyword specifying the file mode: **Append** - Opens a file for sequential writing and sets the file pointer at the end of file. **Binary** - Opens a file for sequential reading and writing.

	Input - Opens file for sequential reading. **Output** - Opens a file for sequential writing. **Update** - Opens a file for sequential reading and writing. Better optimized for (**Rel**)**Seek** then **Binary**. **Random** - Opens a file for random reading and writing. See **Field** for more information. If unspecified, the file is opened for **Random** access.
access	Optional. Keyword specifying the operations permitted on the open file: **Access Read** - Read access only, even when a file is **Lock Write**. **Access Write** - Write access only, even when a file is **Lock Read**. **Access Read Write** - Read/Write access, but file is not accessible when it is locked. **Note: Access** cannot be combined with **For Input**, **For Output**, **and For Update**. For these modes, Access is automatically **Access Read**, **Access Write**, **Access Read Write**, respectively
share	Optional. Keyword specifying the operations restricted on the open file by other processes: **Shared** - Other programs have access. **Lock Read** - Other programs have no Read Access. **Lock Write** - Other programs have no Write Access. **Lock Read Write** - Other programs have no Read or Write Access. This is the default.
Commit	Optional. Writes data to the file immediately without buffering by GFA-BASIC 32 or the

	system.
Based	Optional. **Based** 1 is default. Determines the number of the first record (0 or 1) to be used by **Record**#, **Get**#, and **Put**#.
filenumber	Required. A valid file number in the range 1 to 511, inclusive. Use the **FreeFile** function to obtain the next available file number.
Len	Optional. Number less than or equal to 32,767 (bytes). For files opened for random access, this value is the record length. For sequential files, this value is the number of characters buffered. **Len** <= 1 disables GFA-BASIC 32 buffering, default is 2048 bytes. The **Len** clause is ignored if *mode* is **Binary**.

Example

```
Dim fileno% = FreeFile
Open "C:\TEST.DAT" for Output As # fileno%
' …
Close # fileno%
Open "C:\TEST.DAT" for Input As # fileno%
' …
Close # fileno%
' Open for reading only
Open "C:\TEST.DAT" for Binary Access Read As # 1
' _
Close # 1
' Tidy up
Kill "C:\TEST.DAT"
```

Console: CONIN$ and CONOUT$

There are two reserved pathnames for console input (the keyboard) and console output: "CONIN$" and "CONOUT$".

Initially, standard input, output, and error are assigned to the console. It is possible to use the console regardless of any redirection to these standard devices; just open handles to "CONIN$" or "CONOUT$" using **Open** (*CreateFile*.) Console I/O can then easily be performed with **Input #** and **Print #,** letting GFA-BASIC 32 take responsibility for the correct input.

A process can have only one console at a time. GFA-BASIC 32 applications are GUI programs and are not initialized with a console like DOS-applications. If you need a console (to display status or debugging information), you must first create one. There are two simple parameterless functions for this purpose.

```
Declare Function AllocConsole Lib "kernel32" () As
  Int
Declare Function FreeConsole Lib "kernel32" () As
  Int
```

Before opening "CONIN$" or "CONOUT$" a console must be obtained:

```
Declare Function AllocConsole Lib "kernel32" () As
  Int
Declare Function FreeConsole Lib "kernel32" () As
  Int
Declare Function SetConsoleTitle Lib "kernel32"
  Alias "SetConsoleTitleA" (ByVal lpConsoleTitle As
  String) As Long
Declare Function WriteConsole Lib "kernel32" Alias
  "WriteConsoleA" (ByVal hConsoleOutput As Long,
  lpBuffer As Long, _
  ByVal nNumberOfCharsToWrite As Long,
    lpNumberOfCharsWritten As Long, lpReserved As
    Long) As Long
Dim a$
```

```
If AllocConsole()
  ~SetConsoleTitle("Win32 Console API Demo")
  Open "conout$" for Update As # 1, Len = 1
  Open "conin$" for Input As # 2, Len = 1
  Print # 1, "Test"
  Print # 1, _File(# 1)
  Print # 1, _File(# 2)
  Input # 2, a$
  WriteConsole(_File(# 1), V:a$, Len(a$), Null,
    Null)
  Input # 2, a$
  Close
  ~FreeConsole()
EndIf
```

The **Len** = 1 clause disables the internal buffering of GFA-BASIC 32.

The handle returned from **_File**(#) can be used in the console API functions taking a handle to the console like *WriteConsole* and *ReadConsole.*

Console: StdIn and StdOut

A console process uses handles to access the input and screen buffers of its console. A GUI process must create a console before it can use these standard handles (STDIN, STDOUT, and STDERR). Prevously, these handles had standard values 0, 1, and 2. In Win32 however, the (file) handles must be obtained using the *GetStdHandle*() API function. The return value is a file handle that can be used with API functions for I/O and for console read/write.

GFA-BASIC 32 supports the use of these standard handles without using API functions. Opening a file named "std:" will force GFA-BASIC 32 to use one of the standard handles.

The **For Output** and **For Input** clause determine which standard handle is used.

```
Open "std:" for Input As # 1, Len = 1  //
  STD_INPUT_HANDLE
Open "std:" for Output As # 2, Len = 1 //
  STD_OUTPUT_HANDLE
```

GFA-BASIC 32 redirects the standard devices to its own file handling mechanism. As a result, the normal BASIC I/O commands can be used to access the console.

Note: Regardless of any redirection to these standard devices, the console can still be used by opening handles to "CONIN$" or "CONOUT$".

Remarks

You can specify a hardware port in *pathname*$, although only supported by Windows 95/98/Me. Starting with NT, reading and writing ports using file handles is no longer allowed. The following names are defined:

LPT1:,...LPT4:	parallel port (Centronics)
COM1:,...COM4:	serial port (RS232)
CON:	keyboard/screen

See Also

Close, _File(), Field, Record, RelSeek, Seek, Lof, Eof, Loc

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Close Command

Purpose

Closes a I/O channel.

Syntax

Close [#n]

n:integer expression

Description

Close [#n] closes a channel to a file or peripheral device, previously opened with **Open**. The parameter n contains the number of the channel to close.

If no channel number is given all open file channels are closed.

See Also

Open

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Reset Command

Purpose

Closes all disk files opened using the **Open** statement.

Syntax

Reset

Description

The **Reset** statement closes all active files opened by the **Open** statement and writes the contents of all file buffers to disk.

Example

```
Dim FileNumber%
For FileNumber = 1 To 5    ' Loop 5 times.
  ' Open file for output. FileNumber is
    concatenated into the string
  ' TEST for the file name, but is a number
    following a #.
  Open App.Path & "\TEST" & FileNumber for Output
    As # FileNumber
  Write # FileNumber, "Hello World"    ' Write data
    to file.
Next FileNumber
Reset    ' Close files and write contents to disk.
// Tidy up
For FileNumber = 1 To 5 : Kill App.Path & "\TEST"
  & FileNumber : Next FileNumber
```

Remarks

Close without an argument performs the same action.
Reset is VB compatible.

See Also

Open, Close, Flush, Commit

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Commit Command

Purpose

Flushes a file directly to disk.

Syntax

Commit #file

Description

The **Commit** command forces the GFA-BASIC 32 file buffer to write to the operating system. The operating system writes the data as well. **Commit** ensures that the specified file is flushed immediately, not at the operating system's discretion.

Example

```
Local Int32 n, a(10)
For n = 0 To 10 : a(n) = Rand(10) : Print a(n) :
  Next n
Open "c:\Test.dat" for Output As # 1
BPut # 1, V:a(0), 44
Commit # 1          // Forces the OS to save the
  file to HDD
Close # 1
Print
ArrayFill a(), 0
Open "c:\Test.dat" for Input As # 1
BGet # 1, V:a(0), 44
Close # 1
Kill "c:\Test.dat" // Tidy up line
```

```
For n = 0 To 10 : Print a(n) : Next n
```

Remarks

See Also

Flush, Open

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Flush Command

Purpose

Clears the buffers for this file and causes all buffered data to be written to the file.

Syntax

Flush #n

n: iexp

Description

The command **Flush** writes the contents of a GFA-BASIC 32 file buffer to the file. This does not mean, that the data will be written to disk immediately, the data is buffered by the OS. It will (probably) be transferred through the cable to the other networked computer.

The **Commit** (to-disk) command lets you ensure that critical data is written directly to disk rather than to the operating system buffers.

Example

```
Open App.Path & "\test.sav" for Output As # 1
Print # 1; "Save Data"
Flush # 1
Close # 1
Kill App.Path & "\test.sav" // Tidy-up line
```

Remarks

Flush is automatically called for Lock and Unlock.

See Also

Commit, Open, Lock

{Created by Sjouke Hamstra; Last updated: 06/10/2014 by James Gaite}

Print # Command

Purpose

Writes display-formatted data to a sequential file.

Syntax

Print #n[, y, a$,...]

x, y:aexp
a$:sexp

Description

Print # outputs data to a previously opened channel. n is a channel number in the range from 0 to 511. Other than that, **Print #** is equivalent to **Print**.

Example

```
OpenW 1
Local a$, x%, ch%, i%
a$ = "Writing a file"
Text 0, 20, a$
Open "C:\TEST.DAT" for Output As # 1
Print # 1, "Hallo GFA"
Print # 1, "GFA-"
Print # 1, "BASIC 32"
Close # 1
Text 0, 40, "Press any key"
KeyGet x%
a$ = "Reading a file"
WindGet 14, ch%
```

```
Text 0, 60, a$
Open "C:\TEST.DAT" for Input As # 1
For i% = 1 To 3
  Input # 1, a$
  Text 0, 60 + i% * ch%, a$
Next i%
Close # 1
// Tidy-up line
Kill "c:\TEST.DAT"
```

Opens the file TEST.DAT on drive C and writes the strings Hello GFA, GFA-, and BASIC to it. The file then read back in again.

Remarks

Input # reads a line until the next comma. German numbers are often printed using a comma to separate the fractional part. To prevent problems, write numbers using **Write#**, or change the number format with **Mode Using**.

See Also

Print, Using, Write#, Mode

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Spc and Tab Commands

Purpose

Affects the position of the next output in a **Print** statement.

Syntax

Spc(n)

Tab(n)

_Tab(n)

n:integer expression

Description

Both of these commands can only be used as part of a **Print** statement and not as standalone commands. When included in a **Print** statement, they affect the position where the next string is to be placed in slightly different ways: **Spc** inserts *n* spaces, moving the cursor that many places to the right while overwriting any characters in-between; **Tab** and **_Tab** move the cursor to the column defined by *n*, which means it is possible to move the cursor back before the last printed statement. (The **Tab** command treats the left hand column as column number 1, while **_Tab** treats it as column number 0 - therefore, **_Tab**(9) is equivalent to **Tab**(10).)

Example

```
Local a%
```

```
OpenW 1 : Win_1.FontName = "Courier" :
  Win_1.AutoRedraw = 1
// Prints 'HelloHelloHelloHello' starting at
  column 22
Print Tab(22); "HelloHelloHelloHello";
Text 1, 40, "Press a key" : KeyGet a% // Press a
  key
// Moves cursor back to column 27, overwrites with
  text...
Print Tab(27); " and Goodbye";
Text 1, 40, "And another key..." : KeyGet a% //
  Press a key
// ...and then uses Spc to blank out the remaining
  letters
Print Spc(3);
Text 1, 40, "And yet another..." : KeyGet a% //
  Press a key
// Inserts this text before to complete the
  statement
Print _Tab(9); "I shall say"
Text 1, 40, "And now close the Window"
Do : Sleep : Until Win_1 Is Nothing
```

Note the use of the semi-colon at the end of each statement to keep the text all on one line.

See Also

Locate, VTab, HTab

{Created by Sjouke Hamstra; Last updated: 17/05/2017 by James Gaite}

EOF Function

Purpose

Tests if the data pointer points to the end of a file.

Syntax

EOF(#n)

Description

EOF(#n) always acts on the file on the previously opened channel n, and returns -1 if the data pointer points to the end of this file or 0 if not.

Example

```
Auto a$, i%
OpenW 1 : Win_1.PrintWrap = True
Open App.Path & "\TEST.DAT" for Output As # 1
For i% = 1 To 100
  Print # 1, Str$(i%, 3)`
Next i%
Close # 1
Open App.Path & "\TEST.DAT" for Input As # 1
Do Until EOF(# 1)
  Input # 1, a$
  Print ""; a$`
Loop
Close # 1
Kill App.Path & "\TEST.DAT" // Tidy up line
```

Opens TEST.DAT file in the application folder and reads its contents until the end of file.

Known Issues

In earlier versions, **EOF**() didn't work correctly with text files as "internal resource files" (those files that are included in the source code and which name begins with ":"): **EOF**() was true after reading the first line even if there are many more text-lines in the "internal resource file" (it is the required function of **TextEOF** to test for an end-of-text marker, not **EOF**). As of gfawin23.ocx version 2.341, this issue has been fixed.

The only workaround for this old error was to set up a **Try...Catch** structure around the file processing and use this in conjunction with the 'End of File reached' error message to emulate the function of **EOF**.

See Also

Loc(), Lof(), TextEOF

{Created by Sjouke Hamstra; Last updated: 08/03/2018 by James Gaite}

FileAttr Function

Purpose

Returns a Long representing the GFA-BASIC 32 file mode settings for files opened using the **Open** statement.

Syntax

FileAttr(#file, attr)

file: integer expression (0..511)
attr:iexp

Description

FileAttr returns the I/O settings of the files created or opened with the GFA-BASIC 32 command **Open**.

attr indicates the type of information to return.

FileAttr(#, 1)	file mode: 1 = Input 2 = Output 4 = Random 8 = Append 32 = Binary
FileAttr(#, 2)	the file handle (if necessary for System calls), same as **_File**(#)
FileAttr(#, 3)	the size of the GFA-BASIC 32 file buffer (set with **Len** = n)
FileAttr(#, 4)	Based 1 or Based 0 (set with **Option Base** , 0 | 1)

FileAttr(#, 5)	non-zero (-1): the file is not seekable; you cannot use: **Seek**, **RelSeek**, **Record**, **Lof**, **EOF** etc. (LPT:, CON:).
FileAttr(#, 6)	record size, random files = size of buffer, otherwise 1

Example

```
Open App.Path & "\Test.Dat" for Output As # 1
Print FileAttr( # 1, 1)       // Prints 2
Close # 1
Kill App.Path & "\Test.Dat" // Tidy-up line
```

Remarks

GFA-BASIC 32 manages a file record for each opened file. **FileAttr** allows retrieving the record fields.

GetAttr and **SetAttr** retrieve and set the file type attributes at the system level.

See Also

_File(), FileLen(), GetAttr, SetAttr

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

FreeFile Function

Purpose

Returns the next free file number.

Syntax

n = **FreeFile**

n:iexp

Description

Returns the next free file number to be used with the **Open** statement. The return value is an integer in the range 0 .. 511.

Example

```
Debug.Show
Trace FreeFile
```

Remarks

If working with more complex programs, it is recommend to use a variable, rather than a fixed number.

```
Local Dat% = FreeFile
Open App.Path & "\test.dat" for Output As # Dat%
Close # Dat%
Kill App.Path & "\test.dat" // Tidy-up line
```

See Also

Open

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Loc Function

Purpose

Returns the current position of the file data pointer.

Syntax

large = **Loc**(#n)

% = **Loc%**(#n)

Description

Loc[%](#n) works only on files previously opened with **Open** using channel n (0 <= n <= 511) and returns the current position of the data pointer (locate).

Loc() returns a 64-bit integer and is suited for files sizes of 4 GB.

Loc%() returns a 32-bit integer and is suited for files sizes with a maximum of 2 GB.

Example

```
Local a$, n As Int32
OpenW # 1
Open "c:\TEST.DAT" for Output As # 1
For n = 1 To 7
  Write # 1, Format(n, "dddd")
Next n
Close # 1
Open "C:\TEST.DAT" for Input As # 1
```

```
Do Until EOF(# 1)
  Input # 1, a$
  Print " "; a$, Loc(# 1)
Loop
Close # 1
Kill "c:\test.dat" // Tidy-up Line
```

Opens the file TEST.DAT in current directory andreads its contents as well as the position of the data pointer until end of file.

Remarks

The functions **Loc%**(), **Lof%**(), **Record%**#, **Seek%**#, **RelSeek%**# etc. always use 32 bits integers and are therefore limited to files with a maximum size of 2 GB

See Also

Eof(), Lof(), Record#, Seek#, RelSeek#

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Lof Function

Purpose

Determines the length of a file.

Syntax

large = **Lof**(#n)

long = **Lof%**(#n)

Description

Lof[%](#n) works only on a file previously opened with Open though the channel n and returns its length in bytes (length of file).

Lof() returns a 64-bit integer and is suited for files sizes of 4 GB.

Lof%() returns a 32-bit integer and is suited for files sizes with a maximum size of 2 GB.

Example

```
OpenW # 1
Open "c:\Test.Dat" for Output As # 1
Print # 1, String$(200, "A")
Close # 1
Open  "c:\Test.Dat" for Input As # 1
Print "File length in bytes: "; LOF(# 1) // Prints
 200
Close # 1
```

Opens file TEST.DAT in current directory and returns its size.

Remarks

The functions **Loc%**(), **Lof%**(), **Record%**#, **Seek%**#, **RelSeek%**# etc. always use 32 bits integers and are therefore limited to files with a maximum size of 2 GB. (VB compatibility)

See Also

Eof(), Loc(), Record#, Seek#, RelSeek#

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Seek, RelSeek and SeekEnd Commands

Purpose

Relative positioning of the data pointer

Syntax

Seek[**%**] #n, lpos (command)

RelSeek[**%**] #n, lpos

SeekEnd #n

lpos = **Seek**[**%**](#n) (function)

n:integer expression; channel number
lpos:Large expression (or integer for xxx% commands)

Description

Seek, **RelSeek** and **SeekEnd** enable access to index sequential files with channel numbers from 0 to 511 previously opened with **Open**; they can not be used with peripheral devices.

The **Seek** command moves the file data pointer to the position specfied in the *lpos* value; **RelSeek** performs a similar task but moves the pointer *lpos* places further on (or back if *lpos* is negative - **RelSeek** only) from the pointer's current position. Care should be taken with both these commands not to move the pointer beyond either the start or the end of file as this will result in an error.

SeekEnd has but one task and that is to move the file data pointer to the end of the file.

Finally, the position of the file data pointer can be returned by using the **Seek** function.

With all the above commands and functions, when the suffix % is used, it restricts their use to files no greater than 2GB and returns values as 32-bit integers; these variants are included for compatibility reasons.

Example

```
// Create Test File
Local a$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ", b&
BSave App.Path & "\Test.Dat", V:a$, 26
// Open Test File
Open App.Path & "\test.dat" for Binary As # 1
// Reading a byte moves the pointer one place
  on...
a$ = Chr(Inp|(# 1)) : Print Seek(# 1)
// ...while reading a word moves the pointer two
  places on...
b& = Inp&(# 1) : Print Seek(# 1)
// .. and so on.
// To move to the tenth byte...
Seek # 1, 10 : Print Seek(# 1)
// ...and to move it six bytes further on...
RelSeek # 1, 6 : Print Seek(# 1)
// ...and then two bytes back...
RelSeek # 1, -3 : Print Seek(# 1)
// ...which brings us to position 13 which prints
  N
Print Chr(Inp|(# 1))
// Then, off to the end of the file...
SeekEnd # 1 : Print Seek(# 1)
// ...then back to the beginning using either...
```

```
RelSeek # 1, -26 : Print Seek(# 1)
// ...or...
Seek # 1, 0
// Finally, some changes to the file...
// ... replacing F with an asterisk...
Seek # 1, 6 : Out| # 1, Asc("*")
// ... and P (pos 15) with an underscore...
RelSeek # 1, 8 : Out| # 1, Asc("_")
// ...and then read and print the file contents...
Seek # 1, 0 : Input # 1;a$ : Print a$
//...and then show the pointer
Print Seek(# 1)
Close # 1
Kill App.Path & "\Test.Dat"
```

Remarks

The **Seek** function is synonymous with **Loc**.

Known Issues

Sometimes, **Relseek** does not work well in large files and can cause the file pointer to move to the wrong place. In these circumstances, use `Seek #n, Loc(#n) + bytes` where *n* is the file number and *bytes* is the number of bytes you wish to move. This workaround works for backward moves as well: just replace the plus sign with a minus.

See Also

Seek, SeekEnd, Loc

{Created by Sjouke Hamstra; Last updated: 08/03/2018 by James Gaite}

_File Function

Purpose

Returns the MS-DOS or MS-Windows file handle of the opened file #n. If file #n is not opened a 0 is returned, or in case of devices (LPT1:...) a negative number is returned.

Syntax

x = **_File**(n%)

x: Handle

Description

n% must be in the range between 0 and 511 to correspond to available channel numbers.

Example

```
OpenW # 1
Open "lpt1:" for Output As # 1
Open "test1" for Output As # 2
Open "test2" for Output As # 3
Local i%
For i% = 1 To 5
  Print _File(# i%)
  Close # i%
Next i%
```

Prints the corresponding MS-DOS handles.

See Also

Open

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

TextEOF Function

Purpose

Tests for end-of-file.

Syntax

? = **TextEOF**(#n)

n:iexp

Description

Like **EOF**() this function returns True when the end of file is reached. In addition, this function returns True when the next byte in the stream has value #26 (Control-Z).

TextEOF is required for text files in internal resource files, those files that are included in the source code and which name begins with ":").

Example

The long way of doing it.

```
// Create dummy file
Local a$, a1$ = "This is record 1", a2$ = "This is
  a dummy file", a3$ = #26"This part won't be
  copied"
Open App.Path & "Dummy.Txt" for Binary As # 1
Print # 1; a1$
Print # 1; a2$
'Print # 1; a3$
```

```
Seek # 1, 0
// Copy file up to #26
Open App.Path & "Dummy.Tx2" for Output As # 2
While Not EOF(# 1)
  Line Input # 1, a$
  If EOF(# 1)
    Exit If Left(a$, 1) = #26
    If InStr(a$, #26) <> 0
      a$ = Left$(a$, InStr(a$, #26) - 1)
    EndIf
  EndIf
  Trace a$
  Print # 2, a$
Wend
Close # 1 : Close # 2
// Tidy Up
Kill App.Path & "Dummy.Txt"
Kill App.Path & "Dummy.Tx2"
```

Now with TextEOF...

```
// Create dummy file
Local a$, a1$ = "This is record 1", a2$ = "This is
  a dummy file", a3$ = #26"This part won't be
  copied"
Open App.Path & "Dummy.Txt" for Binary As # 1
Print # 1; a1$
Print # 1; a2$
Print # 1; a3$
Seek # 1, 0
// Copy file up to #26
Open App.Path & "Dummy.Tx2" for Output As # 2
While Not TextEOF(# 1)
  Line Input # 1, a$
  Trace a$
  Print # 2, a$
Wend
```

```
Close # 1
Close # 2
```

Remarks

Input and **Line Input** test for a **TextEOF** as well.

See Also

Eof, Loc, Lof, RecordLOF

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

RecordLOF Function

Purpose

Returns the number of record in a file.

Syntax

nr = **RecordLOF**[**%**](# n)

n:iexp
nr:large or integer exp

Description

RecordLOF returns a **Large** containing the number of records in a random-access file and the number of bytes in normal file.

RecordLOF% returns a 32-bit integer and is only usable for file size < 2GB.

Example

```
Global age%, firstname$, ct|(5), i%, n1$, n2$,
  nr|, secondname$
OpenW # 1
Open App.Path & "\Musicians.DAT" for Random As #
  1, Len = 52
Field # 1, 24 As firstname$, 24 As secondname$, 4
  At(V:age%)
//
For i% = 1 To 5
  Read n1$, n2$, age%
  Lset firstname$ = n1$
  Lset secondname$ = n2$
  Put # 1, i%
  ct|(i%) = i%
Next i%
Close # 1
Data
  Harold,Faltemeyer,56,Robin,Williams,32,Barry,Mani
  low,78,Bryan,Adams,52,Demi,Lovato,21
//
Open App.Path & "\Musicians.DAT" for Random As #
  1, Len = 52
Field # 1, 24 As firstname$, 24 As secondname$, 4
  At(V:age%)
Print "No of Records in File ="; RecordLOF(# 1)
Close # 1
Kill App.Path & "\Musicians.DAT"  // Tidy-up line
```

Remarks

See Also

Lof

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Kill Command

Purpose

Deletes a file.

Syntax

Kill filename$

Kill [Yes | Prompt | Undo | NoUndo | Silent | Files | ,] filename$

filename$:sexp; path name

Description

Kill *filename$* deletes the specified file. Without a path the file is searched in the current directory. When the file isn't found an error is displayed. **Kill** deletes one file only.

Kill can be extended using the same options as **KillFile**/**DeleteFile** and delete complete folders and can use wildcards.

Yes	Disable confirmation dialog box.
Prompt	Inquiry before deleting (default).
Undo	Don't permanently delete file.
NoUndo	The files are deleted irretrievable (default).
Silent	Deletes the file without feedback.
Files	Only files will be deleted, no directories

(**Kill Files** "C:\temp\*" deletes all files in the folder temp but not any subdirectory in temp.) This you can do with **KillFile** "C:\temp\*".

To prevent deleting the file(s) permanently use additional keywords (**Prompt**, **Undo**).

Example

```
Local path$ = "C:\TEST.TXT"
Open path$ for Output As # 1 : Close # 1
If Exist(path$) Then Kill Silent Files path$
```

Checks if the file with the name TEST.TXT exists on drive C and deletes it.

Remarks

Leading commas in front of the file name are ignored, like

Kill Undo, **Files**, **Prompt**,,,, "c:\temp\*"

In contrast with **Kill**, the commands **KillFile** and **DeleteFile** don't delete files permanently by default.

See Also

KillFile, DeleteFile

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

Lock, UnLock

Purpose

Controls access by other processes to all or part of a file opened using the **Open** statement.

Syntax

Lock # n [, recordrange]
Unlock # n[, recordrange]

err **= Lock(#** n [, recordrange] **)**
err = **Unlock(#** n[, recordrange] **)**

n:iexp (0..511)
recordrange:recnumber | [start] ***To*** *end*
err:iexp

Description

The **Lock** and **Unlock** statements/functions are used in environments where several processes might need access to the same file. With the command **Lock** you can lock a part (i.e. one record of a file) of a previously opened file. **Lock** and **Unlock** statements are always used in pairs. The arguments to **Lock** and **Unlock** must match exactly.

The following applies to both **Lock** and **UnLock (**Random files start counting at 1, unless the file was opened with **Based** 0. For sequential files *recnumber* is the byte number.):

Lock #n
locks the entire file.

Lock #n, offset, count
offset is the first byte from which count start to lock till the end of the number of bytes, specified in count.

Lock #n, recnumber
locks a record with the specified number of a random file. Random files start counting at 1, unless the file was opened with **Based** 0. For sequential files *recnumber* is the byte number.

Lock #n, recnumstart **To** recnumend
locks a range of records or bytes.

Lock #n, **To** recnumend
all records from the first record to the end of the range (*end*) are locked (or unlocked).

When a locked file is accessed by another process, the (**Un**)**Lock** commands generate a runtime error. To prevent your application from crashing these statements should be enclosed in a Try/Catch block. The runtime errors are returned as function return values when (**Un**)**Lock** is used as a function.

Example

```
Local a$, ret%
Open "c:\Test.Dat" for Output As # 1
Write # 1, String$(200, "A")
Close # 1
Open "c:\Test.Dat" for Input Shared As # 1
ret% = Lock(# 1, 0, 100) // no runtime error
OpenW 1
If ret% = 0
```

```
  Print "This file has exclusive access"
  Flush # 1
  a$ = Input$(100, # 1)
  Print "Extracted info: "; a$
  Unlock # 1, 0, 100    // now: no error handling!!
  Print "Now everyone can connect again."
Else
  Print "Error #"; ret%; " during locking"
EndIf
Close # 1
```

Remarks

In a multitasking environment often problems arise with simultaneous access of the same file. To solve this problem easily, open a file with the command Open, but without the option **Shared**. This will open the file exclusively, and all other applications have to wait until the file is closed.

Otherwise, to allow multiple applications access to a file it should be opened with the Shared flag. Using (Un)Lock an application can lock the part of the file that it should access and not longer than necessary.

See Also

Open

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

KillFile Command

Purpose

Deletes a file, files, or subdirectories.

Syntax

KillFile filename$

KillFile [Yes | Prompt | Undo | NoUndo | Silent | Files | ,] filename$

filename$:sexp; path name

Description

KillFile filename$ deletes the specified file. Without a path the file is searched in the current directory. When the file isn't found an error is displayed. **KillFile** can delete complete folders and can use wildcards.

KillFile deletes files with the default settings **Prompt** and **Undo**; the files are deleted by moving them to the Recycle Bin. Other keywords are:

Yes	Disable confirmation dialog box.
Prompt	Inquiry before deleting (default).
Undo	Don't permanently delete file.
NoUndo	The files are deleted irretrievable (default).
Silent	Deletes the file without feedback.
Files	Only files will be deleted, no directories

For instance, **KillFile Files** "C:\temp\*" deletes all files in the folder temp but not any subdirectory in temp. This you can do with **KillFile** "C:\temp\*".

Note The wildcard for all files is "*", not "*.*".

Example

```
Local path$ = "C:\TEST.TXT"
Open path$ for Output As # 1 : Close # 1
If Exist(path$) Then KillFile Silent Files path$
```

Remarks

Leading commas in front of the file name are ignored, like

```
KillFile Undo, Files, Prompt, , , , "c:\temp\*"
KillFile Undo, Files, Prompt "c:\temp\*"
```

DeleteFile is a synonym for **KillFile**.

See Also

Kill, DeleteFile

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

DeleteFile Command

Purpose

deletes a file or subdirectories

Syntax

DeleteFile filename$

DeleteFile [Yes | Prompt | Undo | NoUndo | Silent | Files | ,] filename$

filename$:sexp; path name

Description

DeleteFile filename$ deletes the specified file. Without a path the file is searched in the current directory. When the file isn't found an error is displayed. **DeleteFile** can delete complete folders and can use wildcards.

DeleteFile deletes files with the default settings Prompt and Undo. Other keywords are:

Yes	Disable confirmation dialog box.
Prompt	Inquiry before deleting (default).
Undo	Don't permanently delete file.(default).
NoUndo	The files are deleted irretrievable.
Silent	Deletes the file without feedback.
Files	Only files will be deleted, no directories

DeleteFile Files "C:\temp\*" deletes all files in the folder temp but not any subdirectory in temp. This you can do with **DeleteFile** "C:\temp\*".

Example

```
Local path$ = "C:\TEST.TXT"
Open path$ for Output As # 1 : Close # 1
If Exist(path$) Then DeleteFile Silent Files path$
```

Remarks

Leading commas in front of the file name are ignored, like

```
DeleteFile Undo, Files, Prompt, , , , "c:\temp\*"
```

KillFile is a synonym for **DeleteFile**.

See Also

Kill, KillFile

{Created by Sjouke Hamstra; Last updated: 30/09/2014 by James Gaite}

Files Command

Purpose

Prints the directories in the current path name.

Syntax

Files path$ [**To** file$]

path$:sexp; current path name

file$:sexp; optional file name

Description

Files *path*$ returns the contents of the directories in pathname specified in *path*$. If *path*$ ends with a ":" or "\", GFA-BASIC automatically appends "*.*". The default destination for the output of the directory is the screen. Wildcards are allowed.

The specification of **To** *file*$ is optional. It can be used to divert the directory output to a file or a peripheral device.

In contrast to **Dir To** each file in **Files To** is first listed in its MSDOS name (8.3), followed by file size (character position 14 to 24), date and time (position 25 to 44), and ends with the Windows name (position 45).

Example

```
OpenW # 1
FontName = "terminal"
```

```
PrintScroll = 1
Files
```

See Also

Dir

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

Get# Command

Purpose

reads a record from a random access file.

Syntax

Get #n [[,record], varname]

Get% #n [[,record], varname]

n:integer expression; channel number
record:integer expression; record number
varname:variable aexp

Description

Get # reads a record from an Random Access file through the channel n (from 0 to 511), previously opened with **Open**. *record* is an optional parameter and contains a value between 1 and the number of records within the file. If record is not specified the next record in file is always read. Otherwise the record specified in record is read.

The first record or byte in a file is at position 1, the second record or byte is at position 2, and so on. This can be changed using **Option Base** ,n

The second optional parameter *varname* is a variable of any type into which data is read. This syntax allows to use **Get** without a **Field** command, in a VB compatible manner. The length of this variable should be enough to hold a record (Len=).

Get% # reads from a file with a maximum size 2GB.

Example

```
Global city$, i%, n$, name$, o$, postcode%, s$,
  strt$
OpenW # 1
Open App.Path & "\Addresses.DAT" for Random As #
  1, Len = 64
Field # 1, 24 As name$, 24 As strt$, 4
  At(V:postcode%), 12 As city$
//
For i% = 1 To 5
  Input "NAME : ";n$
  Input "Street : ";s$
  Input "Postcode: ";postcode%
  Input "City : ";o$
  Lset name$ = n$
  Lset strt$ = s$
  Lset city$ = o$
  Put # 1, i%
  Cls
Next i%
Close # 1
//
Open App.Path & "\Addresses.DAT" for Random As #
  1, Len = 64
Field # 1, 24 As name$, 24 As strt$, 4
  At(V:postcode%), 12 As city$
//
For i% = 1 To 5
  Get # 1, i%
  Print "Record number: "; Str$(i%, 3)
  Print "NAME : "; name$
  Print "Street : "; strt$
  Print "Postcode: "; postcode%
  Print "City : "; city$
```

```
Next i%
Close # 1
Kill App.Path & "\Addresses.DAT"  // Tidy-up line
```

A channel for the random access file is opened first. Next, the record is divided with Field into: 24 bytes for the name, 24 bytes for the street, four bytes for the postal code and 12 bytes for the city, which all together totals 64 bytes. The For...Next loop writes five records to the file ADDRESSES.DAT on drive C. And finally, these records are read in using **Get** and displayed on the screen again.

Remarks

The functions **Loc%**(), **Lof%**(), **Record%#**, **Seek%#** etc. internally use 32 bits integers and are therefore limited to files with a file size upto 2 GB. The versions without % use 64-bit integers and allow access to larger files.

See Also

Field, Put#IO, Record##

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Input, Form Input and Line Input Commands

Purpose

Allows the input of one or more variables, with or without the prompt.

Syntax

Input ["Text",] x [,y,...]
Input ["Text";] x [,y,...]

Line Input ["Text",] a$ [,b$,...]
Line Input ["Text";] a$ [,b$,...]

Form Input n, var
Form Input n **As** var

Text: any text as prompt
a$, b$: string variable
n : integer
var: variant or string
x, y: any variable type

Description

All the above commands always start from the last cursor position. To define the location where the input should take place, the cursor can be positioned using **Print At** followed by a semicolon, **Locate**, **VTab** or **HTab**.

Both **Line Input** and **Input** contain an optional prompt which is separated from the following variables by a comma or a semicolon and both can receive multiple variables, strings only for **Line Input** or any variable type for **Input**. It is advised that **Line Input** is used for inputting strings as it's entries can contain commas, whereas a string entered with **Input** can not. The maximum input length for strings is 10000 characters and special characters can be entered by typing numbers on the numeric key block while holding down the alternate key <Alt>.

If only one variable is requested, its input must be ended by pressing the <Return> or the <Enter> key. If **Input** contains a list of variables the entry of each individual variable is terminated by pressing the <Return> or the <Enter> key. Any corrections within the variable list are made by using the <Backspace>, <Delete> and <Insert> keys, as well as the cursor keys. Unfortunately, unlike in GFABASIC16, it is no longer possible to separate individual variables in the list with commas and confirm them all with one single press of the <Return> or the <Enter> key (see examples for a workaround).

Form Input differs in that it can only accept a single string (or variant) and that any value input is restricted to the number of letters specified by the *n* integer value.

Example

```
OpenW 1
Local a$, a%, b$
Local Double x,  y
HTab 10 : VTab 9
Print "First Name:"; : Form Input 20 As a$
HTab 10 : VTab 10
Line Input "Surname:", b$
```

```
Print AT(40, 20);
Input "Enter two numbers: ";x, y
Print a$`b$`x`y
Do : Sleep : Until Win_1 Is Nothing
```

In GFABASIC16, it was possible to input a list of variables in one input box by separating them with commas; sadly, this no longer works in GFABASIC32, although it is quite easy to replicate this action, as the following code shows:

```
Local a$, p As Int32, x As Double, y As Double
Input "Enter two values:";a$
p = InStr(a$, ",")
If p = 0 // No commas
  x = Val(a$)
  Input "...and the second value:";y
Else
  x = Val(Left(a$, p - 1))
  y = Val(Mid(a$, p + 1))
EndIf
Print x, y
```

Another possible workaround uses InputBox as shown below:

```
// Courtesy of Factor23
Local Int16 a, b
entree("a,b", *a, *b)

Proc entree(t$, ParamArray p())
  Local h As Hash String, l$, i = LBound(p())
  l$ = InputBox(t$)
  Split h[] = l$, ","
  For i = LBound(p()) To UBound(p())
    DblPoke p(i) , Val(h[% i + 1])
  Next i
  Print t$; " : "; l$
```

```
End Proc
```

Remarks

Input, **Line Input** and **Form Input** date from the days before forms and text boxes and are included for backwards compatibility. Better results can be achieved using either InputBox, OCX Richedit, Prompt and OCX TextBox controls.

LineInput and **Input** can both be used to retrieve data from files - see here

See Also

{Created by Sjouke Hamstra; Last updated: 30/03/2016 by James Gaite}

Input #,Input$, Input? and Line Input

Purpose

Reads data from a previously opened file.

Syntax

Input #n,v1[,v2,...]

Line Input #n,s1[,s2...]

$ = **Input$**(count, #n)
$ = **Input?**(count, #n)

n: integer expression; channel number
count: number of characters
#n : channel number
s1, s2,...: strings
v1,v2,...: any variable type

Description

All these commands read data from a file, accessed with the channel number n (from 0 to 511).

For **Line Input** and **Input**, either individual values or whole variable lists can be read, the latter being separated by commas; **Line Input** is optimised to accept string variables and does not read a mid-line comma as a data separator; **Input** is also capable of reading strings (and sometimes does it better), as well as being suited to reading numeric values; both are restricted to inputting ±1,000 character strings and both internally use TextEOF to test for an end-of-file situation.

The **Input$()** and **Input?()** are synonymous and read the specified number of characters from #n into a string variable which is automatically expanded or contracted to fit. Unlike the **Input** and **Line Input** commands, these functions are only limited to the legal size of a string (roughly 2^{28} characters long); however, also unlike the two commands, when reading the full length of a string written by a **Print #** statement, these functions do not then remove the record separator at the end of the string and so, to read the next record, the separator needs to be cleared by using a dummy Input# call.

Example

```
Local a$, b$, c$, d$, e$, f$, g$, n%, txt$ =
  "'Hello, how are you?', 'I am fine,
  thanks'"#13#10"'That's good to hear.'"
Open App.Path & "\temp.dat" for Output As # 1
Write # 1, txt$
Print # 1, txt$
Print # 1; Len(txt$) : Print # 1, txt$
Close # 1
Open App.Path & "\temp.dat" for Input As # 1
Line Input # 1;a$       // a$ reads the first
  iteration of txt$
Line Input # 1;b$, c$   // but both b$ and c$ are
  required to read the second due to the commas
Close # 1
Print "Line Input:" : Print "a$ = "; a$ : Print
  "b$ = "; b$ : Print "c$ = "; c$ : Print
Open App.Path & "\temp.dat" for Input As # 1
```

```
Input # 1;a$                        // a$ reads the
  first iteration of txt$
Input # 1;b$, c$, d$, e$, f$      // but b$, c$, d$,
  e$ and f$ are required to read the second due to
  the commas
Input # 1;n% : g$ = Input$(n, # 1)
Print "Input:" : Print "a$ = "; a$ : Print "b$ =
  "; b$ : Print "c$ = "; c$
Print "d$ = "; d$ : Print "e$ = "; e$ : Print "f$
  = "; f$ : Print
Print "Input$:" : Print "g$ = "; g$
Print "EOF? = "; EOF(# 1) // There is still the
  record separator remaining at the end of the file
Input # 1;a$
Print "EOF? = "; EOF(# 1) // Now the end of the
  file is reached
Close # 1
Kill App.Path & "\temp.dat"
```

Remarks

None of these functions and commands were implemented to work with a "COM:" port or any other interface, but are built to work only with files and are optimized in that direction. If you want to read in through a different interface, please use the ReadFile() Windows API Function instead.

However, Input and Line Input (without the file number) are able to receive input from the keyboard (see here).

Finally, to deal with large string arrays, it is sometimes better to use

See Also

{Created by Sjouke Hamstra; Last updated: 14/07/2015 by James Gaite}

Record Command

Purpose

Specifies the next record to be read with **Get** # or written with **Put** #.

Syntax

Record[**%**] #n, record

Record[%](#n**)**

n:integer expression; channel number
record:integer expression

Description

The command **Record** #n, record specifies the next record to be read with **Get** # or written with **Put** #.

Record% can be used for file sizes less then 2GB.

The function **Record**() returns the current record or byte number.

Example

```
Global age%, firstname$, ct|(5), i%, n1$, n2$,
  nr|, secondname$
OpenW # 1
Open App.Path & "\Musicians.DAT" for Random As #
  1, Len = 52
Field # 1, 24 As firstname$, 24 As secondname$, 4
  At(V:age%)
```

```
//
For i% = 1 To 5
  Read n1$, n2$, age%
  Lset firstname$ = n1$
  Lset secondname$ = n2$
  Put # 1, i%
  ct|(i%) = i%
Next i%
Close # 1
Data
  Harold,Faltemeyer,56,Robin,Williams,32,Barry,Mani
  low,78,Bryan,Adams,52,Demi,Lovato,21
//
Open App.Path & "\Musicians.DAT" for Random As #
  1, Len = 52
Field # 1, 24 As firstname$, 24 As secondname$, 4
  At(V:age%)
For i% = 5 DownTo 1
  nr| = Rand(i%) + 1
  Record # 1, ct|(nr|)
  Get # 1
  Print "Record" & ct|(nr|) & ": " &
    Trim(firstname$) & " " & Trim(secondname$) & "
    aged" & age%
  Delete ct|(nr)
Next i%
Close # 1
Kill App.Path & "\Musicians.DAT"  // Tidy-up line
```

Remarks

The function Record() is the reverse function to the command Record.

Record(# i) = **Loc**(# i) \ **FileAttr**(# i, 6) + **FileAttr**(# i, 4)

See Also

Field, Get#, Put#, Seek, FileAttr

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Put# Command

Purpose

Writes a record to a random access file.

Syntax

Put[%] #n [[,record], variable]

n:integer expression; channel
record:integer expression
variable:variable name

Description

Put # writes a record to an R-file through the channel n (from 0 to 511), previously opened with **Open**. *record* is an optional parameter and contains a value between 0 or 1 depending on **Option Base**, and the number of records within the file. If record is not specified the next record in file is always written out.

Put #n, *variable* writes the contents of the variable to the file.

Put% internally uses 32-bit access and writes records to a file with a maximum size of 2GB.

Example

See Get #.

Remarks

Put # can only add one record to a file. To add several records to an R-file a loop containing a **Put** # must be created.

See Also

Field, Get#, Record

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Write Command

Purpose

Saves data to sequential files for later read with **Input** #.

Syntax

Write [#n,]a[,a$, b,...]

n:integer expression, channel
a, b:aexp
a$:sexp

Description

The **Write** [#n] command is followed by numerical and string expressions which must be separated by commas. **Write** #n writes these expressions sequentially. The characters are enclosed in quotation marks and commas are generally used as separators.

Note that **Write** can be used to print to the **Form** as well.

Example

```
Local f$ = App.Path + "\Test.Dat", a$, i%
AutoRedraw = 1
Open f$ for Output As # 1
Write # 1, 2 * PI, "Hello GFA", _
  Sin(PI ^ 2 / 4)
Close # 1
Open f$ for Input As # 1
For i% = 1 To 3
```

```
  Input # 1, a$
  Print a$
Next i%
Close # 1
Open f$ for Input As # 1
Print "Format of the file: "; _
  Input?(LOF(# 1), # 1)
Close # 1
// Tidy-up line
Kill f$
```

See Also

Print#, Input#

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Bput Command

Purpose

Fast save of an area of memory to a file.

Syntax

Bput #n, addr, count

n, addr, count:integer expression

Description

An area of memory can be saved to disk (RAM disk, hard disk etc.) using **Bput** (block put) and loaded back in with Bget (block get). The channel #n must be opened first with **Open** names$ **for Output As #**. The integer expression addr contains the start address of the memory to be saved. In addition, count must specify the length of the file.

Example

```
' Save and Load an array
OpenW # 1
Dim a%(999), addr%, b%(200), count%, i%
For i% = 0 To 999
  a%(i%) = Rand(1000)
Next i%
addr% = V:a%(0)
count% = (V:a%(1) - V:a%(0)) * 1000
Open "C:\TEST.DAT" for Output As # 1
BPut # 1, addr%, count%
Close # 1
```

```
Open "C:\TEST.DAT" for Input As # 1
addr% = V:b%(0)
count% = (V:b%(1) - V:b%(0)) * 200
BGet # 1, addr%, count%
Close # 1
Kill "c:\TEST.DAT" // Tidy up line
For i% = 1 To 10
  Print b%(i%)
Next i%
```

Remarks

The saving of memory with **Bput** is similar to **BSave**. In contrast to **BSave**, **Bput** saves the data through a previously opened channel under a previously defined file name.

See Also

Bload, BSave, Bget, Open

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

Bget Command

Purpose

Fast read of files saved with Bput.

Syntax

Bget #n, addr, count

n, addr, count:integer expression

Description

Bget (block get) is used to read files stored with **Bput** (block put). A channel for the file must be opened first with **Open**. *addr* contains the address where in memory the file should be loaded. *count* defines how much data should Bget read from the file.

Example

```
' Save and Load an array
OpenW # 1
Dim a%(999), addr%, b%(200), count%, i%
For i% = 0 To 999
  a%(i%) = Rand(1000)
Next i%
addr% = V:a%(0)
count% = (V:a%(1) - V:a%(0)) * 1000
Open "C:\TEST.DAT" for Output As # 1
BPut # 1, addr%, count%
Close # 1
Open "C:\TEST.DAT" for Input As # 1
```

```
addr% = V:b%(0)
count% = (V:b%(1) - V:b%(0)) * 200
BGet # 1, addr%, count%
Close # 1
Kill "c:\TEST.DAT"
For i% = 1 To 10
  Print b%(i%)
Next i%
```

Reads the first 200 values from the file TEST.DAT on drive C from the address **V:** b%(0) into array b%().

Remarks

Bget and **Bput** can also be used to save and read parts of a file.

See Also

BSave, Bload, Bput, Open

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

BSave Command

Purpose

Fast save of an area of memory to a file.

Syntax

BSave a$, addr, count

a$:sexp; file name

addr, count:integer expression

Description

An area of memory can be saved to disk (RAM disk, hard disk etc.) using **BSave** (block save) and loaded back in with **BLoad** (block load). The integer expression addr contains the start address of the memory to be saved. In addition, count must specify the length of the file a$.

Example

```
OpenW # 1
Local addr%, count%, i%
Dim a%(999), b%(999)
For i% = 0 To 999
  a%(i%) = Rand(1000)
Next i%
addr% = V:a%(0)
count% = (V:a%(1) - V:a%(0)) * 1000
BSave "C:\TEST.DAT", addr%, count%
addr% = V:b%(0)
```

```
count% = (V:b%(1) - V:b%(0)) * 1000
BLoad "C:\TEST.DAT", addr%
Kill "C:\TEST.DAT" // Tidy up line
For i% = 1 To 10
  Print b%(600 + i%)
Next i%
```

Remarks

The saving of files using **BSave** is - depending on the medium - 5 to 10 times faster than with **Open**...**Print**# ...**Close**. Even the memory needed by **BSave** is - depending on the file - up to three times smaller.

BSave and **BLoad** access files in a non-sharing mode.

See Also

Bload, Bput, Bget

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

BLoad Command

Purpose

Fast load of files.

Syntax

BLoad filename$, addr

filename$:sexp; file name

addr:integer expression

Description

BLoad (block load) is used to read the file previously stored with **BSave** (block save). The parameter addr contains the address where in memory the file should be loaded.

Example

```
OpenW # 1
Local addr%, count%, i%
Dim a%(999), b%(999)
For i% = 0 To 999
  a%(i%) = Rand(1000)
Next i%
addr% = V:a%(0)
count% = (V:a%(1) - V:a%(0)) * 1000
BSave "C:\TEST.DAT", addr%, count%
addr% = V:b%(0)
count% = (V:b%(1) - V:b%(0)) * 1000
BLoad "C:\TEST.DAT", addr%
```

```
Kill "C:\TEST.DAT" // Tidy up line
For i% = 1 To 10
  Print b%(600 + i%)
Next i%
```

Remarks

BSave and **BLoad** access the file in a non-sharing mode; they do not work with internal filenames starting with ':'.

See Also

BSave, Bput, Bget

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

Inp(#n) Function

Purpose

Reads one or more bytes from a previously opened file.

Syntax

i = **Inp**(#n)

i = **Inp|**(#n)

i = **Inp&**(#n)

i = **Inp%**(#n)

n: integer expression; channel number

Description

Inp(#n) or **Inp|**() reads a byte from a previously opened file. The numerical expression n contains the channel number (from 0 to 511), with which the file is being accessed.

Inp&(#n) reads 2 bytes (16-bit integer) from a previously opened file.

Inp%(#n) reads 4 bytes (32-bit integer) from a previously opened file.

Example

```
OpenW 1
Local i%, a&, b%
```

```
Open App.Path & "\TEST.DAT" for Output As # 1
For i% = 1 To 50
  Print # 1, Str$(i%, 3)
Next
Close # 1
Open App.Path & "\TEST.DAT" for Input As # 1
For i% = 1 To 20
  a& = Inp|(# 1)    ' or Inp()
  Print a&, Chr(a&)
Next i%
Close # 1
Kill App.Path & "\TEST.DAT" // Tidy-up line
```

opens the file TEST.DAT on drive C and reads in a For...Next loop one byte from this file 20 times and prints the values to the screen.

Remarks

Inp|(#) is synonym with **Inp(#)** and can be used instead.

See Also

Out

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Out # Command

Purpose

Writes a value to an already opened file.

Syntax

Out # n, a [,b,c...]

Out| # n, a [,b,c...]

Out& # n, a [,b,c...]

Out% # n, a [,b,c...]

n:integer expression; channel number

a,b,c...:aexp

Description

Out # n writes one or more bytes to a previously opened file. The numerical expression n contains the channel number (from 0 to 511) used to access the file.

Out| # is synonym with **Out #**. **Out& #** writes a 16-bit integer (word) and **Out% #** writes a 32-bit integer.

Example

```
OpenW 1
Local a%, b&, i%
Open "C:\TEST.DAT" for Output As # 1
For i% = 1 To 20
```

```
  Out& # 1, 128
Next i%
Close # 1
OpenW # 1
Print "The file C:\TEST.DAT was written using only
  Out& #n ()" _
  + "out and will be read back now."
Print
Open "C:\TEST.DAT" for Input As # 1
For i% = 1 To 20
  b& = Inp&(# 1)
  Print b&`
Next i%
Close # 1
// Now use Out|, Out and Out% to produce the same
  result
Open "C:\TEST.DAT" for Output As # 1
For i% = 1 To 5
  Out| # 1, 128
  Out # 1, 0
  Out% # 1, $00800080
  Out& # 1, 128
Next i%
Close # 1
Print : Print
Print "The file C:\TEST.DAT was written using all
  four versions of Out #n ()" _
  + " and will be read back now."
Print
Open "C:\TEST.DAT" for Input As # 1
For i% = 1 To 20
  b& = Inp&(# 1)
  Print b&`
Next i%
Close # 1
// Tidy up line
Kill "c:\test.dat"
```

Opens the file TEST.DAT on drive C and writes the word value 128 to it 20 times from inside a For...Next loop.

See Also

Inp

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Inp(PORT) Function

Purpose

Reads a byte from a port.

Syntax

Inp(**PORT** n)

Description

Inp(**PORT** n) reads a byte from a hardware port register, RTC for example.

Example

This command implies an intimate knowledge of the hardware and is not portable.

Remarks

INP(**^** n), **INP|**(**PORT** n) and **INP|**(**^** n) are synonymous with **INP**(**PORT** n) and can be used instead.

INP&(**^** n) can be used to read a word (two bytes) and **INP%**(**PORT** n) a long (four bytes) from successive port addresses.

See Also

Out(PORT)

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Out Port Command

Purpose

Hardware access. Obsolete.

Syntax

Out Port n, m

n:integer expression; port number

m:integer expression

Description

Out Port n, m writes a byte to a hardware port register, RTC for example.

Under NT, 2000, XP, Vista all hardware access is blocked by the operating system, although it should be possible under 95/98/ME.

Therefore, this command is hardly usable.

Remarks

Out ^ n, m or **Out| Port** n, m or **Out| ^**n, m are synonymous with **Out Port** n, m and can be used instead. **Out& ^**n can be used to write a word (two bytes), **Out% Port** n, m to write a long word (four bytes) to successive port addresses.

See Also

Inp(Port)

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Sub Function

Purpose

Subtracts two numeric (integer) expressions.

Syntax

Sub x, y(command)
% = i **Sub** j)(operator)
% = **Sub**(i, j [,m, ...)(function)

x:any numeric variable
y:any numeric expression
i, j:integer expression

Description

Sub x, y subtracts the expression y from value in variable x.

The operator i **Sub** j and function **Sub**(i, j, ...) return the difference between integer expressions. In case one of the parameters isn't an integer, it is converted to a 32-bit value first (using **CInt**).

Example

```
Debug.Show
Dim b# = 1.5
Trace b# Sub 3          // CInt(b#) - 3 = -1
Trace Sub(b#, 3)        // CInt(b#) - 3 = -1
Sub b#, 3 : Trace b#    // b# = -1.5
b# = 2.5
```

```
Trace b# Sub 3            // CInt(b#) - 3 = -1
Trace Sub(b#, 3)          // CInt(b#) * 3 = -1
Sub b#, 3 : Trace b#      // b# = -0.5
```

Remarks

Although the command **Sub** can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed.

Instead of **Sub** x, y, you can use:

```
x = x - y
x := x - y
x -= y
```

When integer variables are used **Sub** doesn't test for overflow!

The **Add**(), **Sub**(), **Mul**() and **Div**() functions can be mixed freely with each other. For example

```
l% = Sub(5 ^ 3, 4 * 20 + 3)
// ...or...
l% = Sub(5 ^ 3, Add(Mul(4, 20), 3))
```

See Also

+, -, *, /F, \, Add, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

Mod Command, Operator, and Function

Purpose

Calculates the modulo of an integer expression based on a second integer expression.

Syntax

Mod v, y(assignment)
% = i **Mod** j(operator)
% = **Mod**(i, j [,m,...])(function)

v:any numeric variable
y:any number expression
i, j, m:integer expression

Description

Mod v, y calculates the modulo of the value in variable v based on the expression y.

The operator i **Mod** j and the function **Mod**(i, j, ...) return an integer value. In case one of the parameters isn't an integer, it is converted to a 32-bit value first (using **CInt**).

Example

```
Debug.Show
Dim b As Double = 7.1, c%
Trace b Mod 3          // CInt(b) Mod 3 = 1
Trace Mod(b, 3)        // CInt(b) Mod 3 = 1
Trace b : c% = b
' Mod Command requires an integer variable
Mod b, 3 : Trace b     // b = 3 - NOT CORRECT
Mod c%, 3 : Trace c%   // c% = 1 - CORRECT
b = 2
Trace b Mod 3.1         // CInt(b) + CInt(3.1) = 2
Trace Mod(b, 3)         // CInt(b) + 3 = 2
Trace Mod(7, 4, 3)      // 0
```

Known Issues

The **Mod** v, y assignment command doesn't work correctly when v is not an integer.

See Also

Add, Sub, Mul, Div, FMod, Dec, Inc, Pred, ++, --, +=, -=, /= , *=

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

Mul and Mul8 Command, Operator & Functions

Purpose

Multiplies a numeric variable with a numeric expression.

Syntax

Mul x, y(command)
% = i **Mul** j(operator)
% = **Mul**(i, j[,m,...])(function)

l = xl **Mul8** yl(operator)
l = **Mul8(**xl, yl [,zl,…])(function)

x:any numeric variable
y:any numeric expression
i, j, m...:integer expression
l,xl,yl,zl...:large expression

Description

The command **Mul** x, j multiplies the value in the numeric variable x (integer or floating-point) with the expression j. The return value type depends on the type of the variable x.

The operator i **Mul** j and function **Mul**(i, j, …) multiply 32-bit integers and return a 32-bit integer value.

Similarly, the operator i **Mul8** j and function **Mul8**(i, j, …) multiply 64-bit integers and return a 64-bit integer value.

Example

```
Debug.Show
Dim b# = 1.5, c As Large = 8
Trace b#
Trace b# Mul 3            // CInt(b#) * 3 = 6
Trace Mul(b#, 3)          // CInt(b#) * 3 = 6
Mul b#, 3 : Trace b#      // b# = 4.5
b# = 2.5 : Trace b#
Trace b# Mul 3            // CInt(b#) * 3 = 6
Trace Mul(b#, 3)          // CInt(b#) * 3 = 6
Mul b#, 3 : Trace b#      // b# = 7.5
Trace c
Trace c Mul8 3            // 24
Trace Mul8(c, 3)          // 24
Mul c, 3 : Trace c        // 24
```

Remarks

Although the command **Mul** can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed.

Instead of **Mul** x, y, you can use:

```
x = x * y
x := x * y
x *= y
```

When integer variables are used **Mul** doesn't test for overflow!

See Also

+, -, *, /F, \, Add, Sub, Mod, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

Div Command

Purpose

Divides a numeric variable by a numeric expression.

Syntax

Div x, y(command)

% = i **Div** j)(operator)

% = **Div**(i, j)(function)

x:numeric variable
y:any numeric expression
i, j:integer expression

Description

Div x, y divides the expression y into the value in variable x. It depends on the type of the variable x which whether the division is an integer or a floating-point division.

The operator i **Div** j and function **Div**(i, j, ...) return an integer value. In case one of the parameters isn't an integer, it is converted to a 32-bit values first (using **CInt**).

Example

```
Debug.Show
Dim b# = 7.5
Trace b# Div 3            // CInt(b#) \ 3 = 2
Trace Div(b#, 3)          // CInt(b#) \ 3 = 2
```

```
Div b#, 3 : Trace b#     // b# = 2.5
b# = 8.5
Trace b# Div 3           // CInt(b#) \ 3 = 2
Trace Div(b#, 3)         // CInt(b#) \ 3 = 2
Div b#, 3 : Trace b#     // b# = 2.833333333333
```

Remarks

The following can be used instead of **Div** x, y:

```
x = x / y
x := x / y
x /= y
```

See Also

+, -, *, /F, \, Add, Sub, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

Fmod Operator

Purpose

Calculates the modulo of a floating point expression based on a second floating point or integer expression.

Syntax

fp = f **Fmod** x

fp, f:floating-point exp
x:any numeric exp

Description

Fmod calculates the modulo of a floating point expression based on a second floating point or an integer expression.

Example

```
Debug.Show
Local vDbl As Double = 142.8544
Trace Mod(142.8544, 15)
Trace 142.8544 Fmod 15
Trace vDbl
Trace vDbl Fmod 15
Trace 142 Fmod 2.6
```

Remarks

The assignment command **Mod** v, y calculates the modulo of the value in variable v based on the expression y. The **Mod** v, y assignment command doesn't work correctly when v is not an integer. To work with floating-point variables use v = v **Fmod** y.

Note The operator i **Mod** j and the function **Mod**(i, j, …) return an integer value. In case one of the parameters isn't an integer, it is converted to a 32-bit value first (using **CInt**).

See Also

Add, Sub, Mul, Div, Mod, Dec, Inc, Pred, ++, --, +=, -=, /=, *=

{Created by Sjouke Hamstra; Last updated: 06/10/2014 by James Gaite}

Sin Function

Purpose

Returns the sine of a numeric expression.

Syntax

# = **Sin**(x)

x:aexp; angle in radians

Description

The sine of an angle in a right-angled triangle corresponds to a quotient between the hypotenuse and the side opposite the angle.

Example

```
Debug.Show
Trace Sin(0)              // Prints 0
Trace Sin(PI / 2)         // Prints 1
Trace Sin(PI)             // Prints
  1.22460635382238e-16 (== 0)
Trace Sin(3 * PI / 2)     // Prints -1
Trace Sin(2 * PI)         // Prints
  -2.44921270764475e-16 (== 0)
```

Remarks

Sin() is the reverse function of **ASin**().

See Also

SinQ(), Cos(), CosQ(), Tan(), ASin(), ACos(), Atn(), ATan()

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Cos Function

Purpose

Returns the cosine of a numeric expression.

Syntax

Cos(x)

x:aexp; angle in radians

Description

The cosine of an angle in a right-angled triangle corresponds to a quotient between the hypotenuse and the side forming the angle. When calculating **Cos**(x) it is assumed that the value of x is given in radians.

Example

```
Debug.Show
Trace Cos(0)          // Prints 1
Trace Cos(PI / 2)     // Prints 6.12303176911189e-
  17
Trace Cos(PI)         // Prints -1
Trace Cos(3 * PI / 2) // Prints
  -1.83690953073357e-16
```

Remarks

Cos() is the reverse function of **ACos**().

See Also

Sin(), SinQ(), CosQ(), Tan(), ASin(), ACos(), Atn(), Atan()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Tan Function

Purpose

Returns the tangent of a numeric expression.

Syntax

*# = **Tan**(x)*

x:aexp; angle in radians

Description

The tangent of an angle corresponds to the quotient of two short sides in a right-angled triangle. The value of x is given in radians.

Example

```
Debug.Show
Trace Tan(PI / 4)          // Prints 1
Trace Tan(PI)              // Prints
  -1.22460635382238e-16 (== 0)
```

Remarks

Tan() is the reverse function of **Atn**() or **Atan**().

See Also

Atn(), Atan() Cos(), CosQ(),(), ASin(), ACos(), Sin(), SinQ()

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

Atn Function

Purpose

Returns the arc tangent of a numeric expression.

Syntax

Atn(x)

x:aexp

Description

Atn(x) expects as function argument x the quotient between the two short sides in a right-angled triangle and returns the angle in radians.

Example

```
OpenW # 1
Print Atn(-PI)          // Prints -1.26...
Print Atn(1)            // Prints 0.78...
Print Atn(PI / 4)       // Prints 0.66...
Print Atn(Tan(PI / 4))  // Prints 0.78...
```

Remarks

Atn() is the reverse function of **Tan**(). **Atn**() is synonymous with **ATan**() and can be used instead.

See Also

Sin(), SinQ(), Cos(), CosQ(), Tan(), Acos(), Atn(), Atan(), Atan2()

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

SinQ Function

Purpose

Returns the extrapolated sine of a numeric expression.

Syntax

# = **SinQ**(x)

x:aexp; angle in degrees

Description

For **SinQ**() GFA-BASIC 32 uses an internal table with sine values in one degree steps. **SinQ**(x) expects, therefore, the expression x to be in degrees. The intermediate values of function x are extrapolated in 1/16- degree steps. This accuracy is sufficient for plotting of graphs on the screen, particularly when there is no co-processor, since this function is several times faster than **Sin**(x).

Example

```
Debug.Show
Trace SinQ(0)   // Prints 0
Trace SinQ(90)  // Prints 1
Trace SinQ(180) // Prints 1.22460635382238e-16 (==
  0)
Trace Sin(PI)   // Prints 1.22460635382238e-16 (==
  0)
Trace SinQ(270) // Prints -1
Trace SinQ(360) // Prints -2.44921270764475e-16
  (== 0)
```

See Also

Sin(), Cos(), CosQ(), Tan(), ASin(), ACos(), Atn(), ATan()

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

CosQ Function

Purpose

Returns the interpolated cosine of a numeric expression.

Syntax

CosQ(x)

x:aexp; angle in degrees

Description

For **CosQ**() GFA-BASIC uses an internal table with cosine values in one degree steps. **CosQ**(x) expects, therefore, the expression x to be in degrees. The intermediate values of function x are interpolated in 1/16- degree steps. This accuracy is sufficient for plotting of graphs on the screen, particularly when there is no co-processor, since this function is several times faster than **Cos**(x).

Example

```
Debug.Show
Trace CosQ(180) // Prints -1
Trace Cos(PI)   // Prints -1
```

See Also

Sin(), SinQ(), Cos(), Tan(), ASin(), ACos(), Atn(), Atan()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

ACos() Trigonometrical Function

Purpose

Returns the arc cosine of a numeric expression.

Syntax

ACos(x)

Description

ACos(x) expects as function argument x the quotient between hypotenuse and the side forming the angle (in a right-angled triangle) and returns the angle in radians. It follows, therefore, that the value of x ranges between -1 (equivalent to **Cos**(PI)) and 1 (equivalent to **Cos**(0)).

Example

```
OpenW # 1
Print Acos(-1)      // Prints 3.14...
Print Acos(0)       // Prints 1.57...
Print Acos(1)       // Prints 0
Print Acos(Cos(PI)) // Prints 3.14...
```

Remarks

ACos() is the reverse function of **Cos**().

See Also

Sin(), Cos(), SinQ(), CosQ() Tan(), ASin(), Atn(), ATan()

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

ATan Function

Purpose

Returns the arc tangent of a numeric expression.

Syntax

ATan(x)

Description

ATan(x) expects as function argument x the quotient between the two short sides in a right-angled triangle and returns the angle in radians.

Example

```
OpenW 1
Print Atan(90)
```

Remarks

ATan() is the reverse function of **Tan**().

Atn() is synonymous with **ATan**() and can be used instead.

See Also

Sin(), SinQ(), Cos(), CosQ(), Tan(), Acos(), Atn(), Atan(), Atan2()

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

ATan2 Function

Purpose

Returns the arc tangent of the quotient of two numeric expressions.

Syntax

ATan2(x, y)

Description

ATan2 returns the arc tangent the quotient of two numeric expressions, without the explicit calculation of the quotient, in case, that a division by zero will not possible. In the contrast to the function **ATan** the results of the function **Atan2** can be placed in all as 0 in all 4 square of the system of coordinates. This will be possible because there will be a way to difference between x > 0 and y < 0 just as x < 0 and y > 0, etc.. With the division of two numeric characters the information, which of both parameters was < 0 are gone (or lost). **ATan**(x) real will be **Atan2**(x,1) and not **Atan2**((-x,-1).

Example

```
OpenW 1
Local x%
Print Atan2(7, 24)
' is the same as:
Print Atan(7 / 24)
'or
Print Atan2(7 / 24, 1)
```

```
KeyGet x%
CloseW # 1
```

Remarks

Converts rectangular coordinates (b, a) to polar (r, theta).

See Also

Sin(), SinQ(), Cos(), CosQ(), Tan(), Acos(), Atn(), Atan(), Atan2()

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

_hypot Function

Purpose

Calculates the hypotenuse.

Syntax

_hypot(x,y)

x,y: aexp

The **_hypot** function calculates the length of the hypotenuse of a right triangle, given the length of the two sides *x* and *y*. A call to **_hypot** is equivalent to the square root of $x^2 + y^2$.

Example

```
OpenW 1
Global a#
a# = _hypot(5, 6)
Print a#    // Result: 7.8102496.....
Do
  Sleep
Until Me Is Nothing
CloseW 1
```

Remarks

An example to use it for: to convert Cartesian coordinates (normal rectangle coordinates) into polar coordinates: angle = **Atan2**(x, y) : radius = **_hypot**(x, y)

See Also

Tanh(), ArSinH(), ArCosH(), ArTanH(), Deg(), Rad(), Sin(), SinQ(), Cos(), CosQ(), Tan(), ASin(), ACos(), ATn(), Atan()

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

Exp, Exp2 and Exp10 Functions

Purpose

Returns the Euler's number e (= 2.178...) to the power of a numeric expression.

Syntax

Exp(x)

Exp2(x)

Exp10(x)

Description

Exp(x) calculates the x-th power of Euler's number e = 2.178...., **Exp2**(x) calculates 2 ^ x, while **Exp10**(x) calculates 10 ^ x.

In all of these functions, *x* can be positive, negative or zero.

Example

```
Debug.Show
Trace Exp(Sqr(2))
Trace Exp2(8)
Trace Exp10(5)
```

Remarks

Exp(x) is the reverse function of **Log**(x), which means:

Exp(**Log**(**PI**)) = **PI** = 3.14...

Similarly, **Log2**(x) and **Log10**(x) are the reverse functions of **Exp2**(x) and **Exp10**(x) respectively.

```
OpenW 1
Local a% = 4
a% = Exp(a%) : Print "Exp(a%) = "; a%
a% = Log(a%) : Print "Log(a%) = "; a%
a% = Exp2(a%) : Print "Exp2(a%) = "; a%
a% = Log2(a%) : Print "Log2(a%) = "; a%
a% = Exp10(a%) : Print "Exp10(a%) = "; a%
a% = Log10(a%) : Print "Log10(a%) = "; a%
```
<**NOTE:** Assigning too high - such as **Exp2(10000)** will result in an overrun error; assigning too low a number - such as **Exp2(-10000)** - will result in an inaccurate result: in the latter example, zero is returned.

See Also

Log(), Log2(), Log10()

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

Log, Log2 and Log10 Numeric Functions

Purpose

Returns a natural, base 2 or base10 logarithm of a numeric expression.

Syntax

Log(x)

Log2(x)

Log10(x)

Description

Log(x) calculates the logarithm of x to the base of Euler's number e (= 2.178....), **Log2**(x) to the base of 2 and **Log10**(x) to the base of 10.

Example

```
Debug.Show
Trace Log(Sqr(2)) // Prints 0.34657
Trace Log2(42)    // Prints 5.392...
Trace Log10(100)  // Prints 2
```

Remarks

Log(x) is the reverse function of Exp(x), which means:

Log(Exp(PI)) = PI = 3.14....

Similarly, **Log2**(x) and **Log10**(x) are the reverse functions of **Exp2**(x) and **Exp10**(x) respectively.

The following function is used to calculate the logarithm of any base:

```
Print LogBasis(8, 2)

Function LogBasis(x, LogBase)
 Return Log(x) / Log(LogBase)
EndFunc
```

See Also

Exp(), Exp2, Exp10

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Sqr, Sqrt Function

Purpose

Returns the positive square root of a numeric expression.

Syntax

# = **Sqr**(x)

# = **Sqrt**(x)

x:aexp

Description

Sqr and **Sqrt** are synonyms and calculate the square root of x.

Example

```
OpenW # 1
Print Sqrt(16)          // prints 4
Print Sqr(PI * 5.3 + 1) // prints 4.20124...
```

Remarks

If the function argument x is less than 0, **Sqr**(x) reports an error.

See Also

Square

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Square Function

Purpose

Returns the square of a numeric expression.

Syntax

# = **Square**(x)

x:aexp

Description

Square multiplies a numeric expression with itself.

Example

```
Debug.Show
Trace Square(FRound(4 * 4 + 4 / 8))
Trace Square(-5.5)
```

Remarks

See Also

Sqr

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Pow Function

Purpose

Returns the value of a base expression taken to a specified power.

Syntax

Pow(x, y)

x, y: aexp

Description

Pow(x, y) is the same as x^y.

Example

```
OpenW 1
Local a%, b%, c%, d%, y%, x%, e%
x% = 2, y% = 5
a% = x ^ y
b% = Pow(x, y)
c% = Exp(y * Log(x))
d% = Exp2(y * Log2(x))
e% = Exp10(y * Log10(x))
Print a%, b%, c%, d%, e%
```

See Also

^

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

_y0(),_y1(),_yn(),_j0(),_j1( ),_jn Functions

Purpose

Compute the Bessel-function

Syntax

Double = _**y0**(x As Double)

Double = _**y1**(x As Double)

Double = _**yn**(n As Int, x As Double)

Double = _**j0**(x As Double)

Double = _**j1**(x As Double)

Double = _**jn**(n As Int, x As Double)

Description

The Bessel functions are commonly used in the mathematics of electromagnetic wave theory.

The **_y0**, **_y1**, and **_yn** routines return Bessel functions of the second kind: orders 0, 1, and n, respectively.

The **_j0**, **_j1**, and **_jn** routines return Bessel functions of the first kind: orders 0, 1, and n, respectively.

Example

```
Print _y0(0.2) // Same as _yn(0, 0.2)
Print _y1(0.2) // Same as _yn(1, 0.2)
Print _yn(2, 0.2)
// .. and so on
Print _j0(0.5) // Same as _jn(0, 0.5)
Print _j1(0.5) // Same as _jn(1, 0.5)
Print _jn(2, 0.5)
// .. and so on
```

See Also

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

LdExp Function

Purpose

Computes a real number from the mantissa and exponent.

Syntax

int **= LdExp**(x, exp)

x: double expression
exp: iexp

Description

LdExp(x, exp) computes a real number from the mantissa and exponent. It is part of a set of three functions, **GetExp**(), **LdExp**() and **Mant**(), that break down a floating-point value.

The **LdExp** function returns the value of *x* ***** 2exp if successful.

The **GetExp**() and **Mant**() correspond to the *frexp* C-function, which breaks down the floating-point value (*exp*) into a mantissa (*m*) and an exponent (*n*), such that the absolute value of *m* is greater than or equal to 0.5 and less than 1.0, and *x* = *m**2n. The integer exponent *n* is obtained using **GetExp**() and m with **Mant**().

Example

```
OpenW 1
Local Double a, b, i, c, x
```

```
Print GetExp(197)
a = GetExp(197)
Print Mant(197)
b = Mant(197)
c = LdExp(a, b)
Print c                          // prints 197
x = 111
Print 2 ^ GetExp(x) * Mant(x)    // prints 111
```

Remarks

See Also

GetExp(), Mant()

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

Mant()

Purpose

Determines the mantissa of a floating-point value

Syntax

int **= Mant**(fexp)

Description

Mant() determines the mantissa of a floating point value. It is part of a set of three functions, **GetExp**(), **LdExp**() and **Mant**(), that break down a floating-point value.

The **GetExp**() and **Mant**() correspond to the **frexp** C-function, which breaks down the floating-point value (*fexp*) into a mantissa (*m*) and an exponent (*n*), such that the absolute value of *m* is greater than or equal to 0.5 and less than 1.0, and *x* = *m**2n. The integer exponent *n* is obtained using **GetExp**() and m with **Mant**().

LdExp(m, exp) computes a real number from the mantissa and exponent.

Example

```
OpenW 1
Local Double a, b, i, c, x
Print GetExp(197)                // Prints 8
a = GetExp(197)
Print Mant(197)                  // Prints
  0.76953125
```

```
b = Mant(197)
c = LdExp(a, b)
Print c                             // Prints 197
x = 111
Print 2 ^ GetExp(x) * Mant(x)       // Prints 111
```

Remarks

See Also

LdExp(), GetExp()

{Created by Sjouke Hamstra; Last updated: 13/10/2014 by James Gaite}

GetExp Function

Purpose

Determines the exponent of the base of two

Syntax

int **= GetExp**(exp)

exp:floating-point expresssion

Description

GetExp() determines the exponent of the base of two. It is part of a set of three functions, **GetExp**(), **LdExp**() and **Mant**(), that break down a floating-point value.

The **GetExp**() and **Mant**() correspond to the **frexp** C-function, which breaks down the floating-point value (*exp*) into a mantissa (*m*) and an exponent (*n*), such that the absolute value of *m* is greater than or equal to 0.5 and less than 1.0, and *x* = *m**2n. The integer exponent *n* is obtained using **GetExp**() and m with **Mant**().

LdExp(m, exp) computes a real number from the mantissa and exponent.

Example

```
Debug.Show
Local Double a, b, i, c, x
Trace GetExp(197)                 // Prints 8
a = GetExp(197)
```

```
Trace Mant(197)                     // Prints
  0.76953125
b = Mant(197)
c = LdExp(a, b)
Trace c                             // Prints 197
x = 111
Trace 2 ^ GetExp(x) * Mant(x)       // Prints 111
```

Remarks

See Also

LdExp(), Mant()

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Randomize Command

Purpose

Seeds the random number generators.

Syntax

Randomize [n]

Description

Randomize [n] seeds the random number generators with the value n. If the random number generator is seeded several times with the same n <> 0, the same sequence of "random numbers" is generated.

Every time a program is run the random number generators are seeded with a "random" number. Therefore, if **Randomize** is not used, each program run will result in **Rnd**, **Random,** or **Rand** producing different random numbers.

Randomize (without parameters) or **Randomize** 0 seeds the random number generator with the value of **Timer**, a random number just like when a program starts up.

See Also

Rnd, Rand, Random

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Rnd Function

Purpose

Generates a random number between 0 (inclusive) and 1 (exclusive).

Syntax

r# = **Rnd**[(x)]

r! = **Rnd!**[(x)]

r#:Double expression
r!:Single expression
x:aexp

Description

The parameter x is optional and has the effect described below. The result of **Rnd** is a **Double**. The result of **Rnd!** is a **Single**.

Rnd(0) returns tha last random number, **Rnd**(positive number) returns, like **Rnd**, a new random number. **Rnd**(negative number) returns always the same random number and executes **Randomize** negative number.

Example

```
Local i%
OpenW # 1
For i% = 1 To 10
  Print Rnd
```

```
Next i%
```

Prints a random number between 0 and 1.

See Also

Random, Rand, Randomize

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

_rand Function

Purpose

Generates a 32-bit integer pseudo random number.

Syntax

x = **_rand**[()]

x : ivar

Description

_**rand** returns a random integer value between 0 and 32767.

_rand and its seed function **_srand** are C-compatible functions and are offered as an alternative to **Rnd**, **Rand**, **Random** and their seed function **Randomize**.

Example

```
Global x%
x = _rand()
MsgBox x
```

Remarks

To create a random value GFA-BASIC 32 uses the 'C'-randomizer, which doesn't need the **Randomize** command to initialize the generator, but instead uses _**srand**.

_**rand**() is faster than **Rand**(), but doesn't have the longer period as **Rand**().

See Also

_rand, _srand, Rand, Randomize, Random, Rnd

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

_srand Function

Purpose

Sets a random starting point.

Syntax

_**srand**(seed)

seed : ivar

Description

The C-compatible _**srand** function sets the starting point for generating a series of pseudorandom integers. To reinitialize the generator, use 1 as the seed argument. Any other value for *seed* sets the generator to a random starting point. **_rand** retrieves the pseudorandom numbers that are generated. Calling _**rand** before any call to _**srand** generates the same sequence as calling **_srand** with seed passed as 1.

_srand(**qTimer**) or **_srand**(**oTimer**) give good random starting values.

Example

```
Global x
~_srand(oTimer) // seed
x = _rand()
MsgBox x
```

Remarks

To create a random value GFA-BASIC 32 uses the 'C'-randomizer, which doesn't need the **Randomize** command to initialize the generator, but instead uses _**srand**.

_**rand**() is faster than **Rand**(), but doesn't have the longer period as **Rand**().

See Also

_rand, _srand, Rand, Randomize, Random, Rnd, qTimer, oTimer

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Abs() Numeric Function

Purpose

Returns the absolute value of a numeric expression.

Syntax

Abs(x)

x:aexp

Description

The number argument can be any valid numeric expression. The return value has the same type as the argument x.

Example

```
Print Abs(-210)    // Prints 210
Print Abs(5 - 10) // Prints 5
Print Abs(-0.3)    // Prints 0.3
```

Remarks

The returned value from **Abs**() depends on the sign of the x argument:

for x < 0 returns -x,

for x = 0 returns 0 and

for x > 0 returns x.

See Also

Sgn

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Sgn Function

Purpose

Returns the sign of a numeric expression.

Syntax

iexp = **Sgn**(x)

x:aexp

Example

```
Debug.Show
Trace Sgn(-210)         // Prints -1
Trace Sgn(Abs(5 - 10))  // Prints 1
Trace Sgn(0)            // Prints 0
```

Remarks

The value returned by the Sgn() function depends on the sign of the argument x:

x < 0 returns -1,

x = 0 returns 0 and

x > 0 returns 1.

See Also

Abs()

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Variat Function

Purpose

Returns the number of permutations of n elements to k-th order without repetition.

Syntax

# = **Variat**(n, k)

Description

Variat(n,k) is defined as:

Variat(n,k)=n!/(n-k)!.

Example

```
OpenW # 1
Print Variat(6, 2)  // Prints 30
```

Remarks

If k > n an error is reported.

See Also

Fact, Combin, Permut

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Combin Function

Purpose

Returns the number of combinations of n elements to k-th order without repetition.

Syntax

Combin(n, k)

Description

Combin(n, k) is defined as: **Combin**(n, k)=n!/((n-k)!*k!)

Example

```
OpenW # 1
Print Combin(6, 2) // Prints 15
```

Remarks

When k > n an error is reported.

See Also

Fact(), Variat()

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Permut Function

Purpose

Returns the number of permutations of n elements to k-th order without repetition.

Syntax

Permut(n, k)

n,k:integer expression

Description

Permut is defined as

Permut(n,k) = n!/(n-k)!.

Example

```
OpenW # 1
Print Permut(6, 2) // Prints 30
```

Remarks

If k > n an error is reported.

See Also

Fact(), Combin(), Variat

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Round, FRound and QRound Functions

Purpose

Rounding the numeric expression x.

Syntax

f = **Round**(x [,n])
f = **FRound**(x [,n])
f = **QRound**(x [, n])

f	*: floating point variable*
x	*: any numeric variable*
n	*: integer*

Description

In all aspects of operation, **Round** and **FRound** are identical: when the optional parameter *n* is omitted, they round *x* to the nearest whole integer, with the decimal 0.5 rounded up; where *n* is positive, they round to *n* decimal points, usually rounding up if the next decimal is a '5', but sometimes rounding down (see example); and where *n* is negative, to the nearest integer multiple of 10^{-n}, rounded up once again if the next digit is a '5'.

QRound uses the 80x87 coprocessor instruction for rounding and, thus, acts like so: when the optional parameter *n* is omitted, it rounds *x* to the nearest EVEN integer; where *n* is positive, it acts like **Round** and rounds to *n* decimal points, usually rounding up if the next decimal is a '5', but sometimes rounding down (see example); and where *n* is negative, to the nearest integer multiple of 10^{-n} where the pertinent digit is EVEN.

The differences are illustrated in the example below (remember **Round** acts in the same way as **FRound**).

Example

```
Debug.Show
Trace QRound(100.5)       // Output: 100
Trace FRound(100.5)       // Output: 101
Debug
Trace QRound(101.5)       // Output: 102
Trace FRound(101.5)       // Output: 102
Debug
Trace QRound(100.55, 1)   // Output: 100.5 (next 5
  rounded down)
Trace FRound(100.55, 1)   // Output: 100.5 (next 5
  rounded down)
Debug
Trace QRound(100.555, 2) // Output: 101.5 (next 5
  rounded up)
Trace FRound(100.555, 2) // Output: 101.5 (next 5
  rounded up)
Debug
Trace QRound(105, -1)     // Output: 100
Trace FRound(105, -1)     // Output: 110
Debug
Trace QRound(115, -1)     // Output: 120
Trace FRound(115, -1)     // Output: 120
```

Remarks

The behaviour of **Round**/**FRound** when dealing with rounding to decimal places is odd and inconsistent: it should follow the pattern set and round up if the next digit is a '5', which it usually does, but not always. To get around this problem, you can use the following rather complicated workaround below to ensure these functions always round up in this situation:

```
Debug.Show
Local Int32 n = 1
Trace FRound(100.55, n)                          //
  Output: 100.5 (next 5 rounded down)
Trace FRound(100.55 + (1 * 10 ^ -(n + 1)), n)   //
  Output: 100.6 (next 5 rounded up)
```

See Also

Ceil(), CInt, Frac(), Fix(), Floor(), Int(), Trunc()

{Created by Sjouke Hamstra; Last updated: 05/08/2019 by James Gaite}

Max and Min Functions

Purpose

These functions return the highest or lowest value among their parameters.

Syntax

int = **iMax** | **MaxI**(i1,i2 [,i3,..., in])

int = **iMin** | **MinI**(i1,i2 [,i3,..., in])

double = **Max**(x1,x2 [,x3,..., xn])

$ = **Min**(x1$,x2$ [,x3$,..., xn$])

$ = **Max**(x1$,x2$ [,x3$,..., xn$])

double = **Min**(x1,x2 [,x3,..., xn])

currency = **MaxCur**(c1,c2 [,c3,..., cn])

currency = **MinCur**(c1,c2 [,c3,..., cn])

int64 = **MaxLarge**(i1,i2 [,i3,..., in])

int64 = **MinLarge**(i1,i2 [,i3,..., in])

c1,c2,...:currency value i1,i2,...:integer (32- or 64-bit) value
x1,x2,...:numerical expression
x1$,x2$,...:numerical expression

Description

Max() return the highest and **Min**() the lowest in a series of numbers or string values given as parameters.

iMax() and **iMin**() do the same but with 32-bit integers (**MaxLarge** and **MinLarge** for 64-bit integers) and are therefore faster; **MaxI** and **MinI** are synonymous with **iMax** and **iMIn** respectively.

Finally, **MaxCur** and **MinCur** return the highest and lowest values from a list of currencies; integers, single and double values can also be used but Variants and Strings will cause errors.

Example

```
Local a% = 5, b# = 5.4
Debug.Show
Trace Max(1, a%, b#, 0.9)
Trace Min(1, a%, b#, 0.9)
Trace iMax(1, a%, b#, 0.9)
Trace iMin(1, a%, b#, 0.9) // 0.9 is rounded up to
  1
```

An example with Currency values:

```
Debug.Show
Local a# = 7.45, a@ = 7.45, b@ = 2.45
Trace MaxCur(3.50, a#, b@)
Trace MaxCur(3.50, CCur(a#), b@)
Trace MinCur(3.50, a@, b@)
```

And an example with String values:

```
Debug.Show
Trace Max("ABC", "BBC", "ABX", "BD")
Trace Min("ABC", "BBC", "ABX", "BD")
```

Remarks

The integer functions (**iMax**, etc) round non-integer parameters using **CInt()** which rounds them to the nearest whole number EXCEPT with decimals of n.5 which it rounds to the nearest even number: therefore, both 3.5 and 4.5 will be rounded to 4, as shown by the example below:

For fastest performance it is advisable to adhere strictly to the variable type particular to the function. Variants and Strings should only be used with **Max** and **Min**.

```
Debug.Show
Trace iMin(3.5, 4.5)
Trace MinI(3.5, 4.5)
Trace iMax(3.5, 4.5)
Trace MaxI(3.5, 4.5)
```

See Also

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Even, Odd Functions

Purpose

Even tests if a numeric expression is even and returns -1 (true) if it is, or 0 if the expression is odd, while **Odd** tests if a numeric expression is odd and returns -1 (true) if it is, or 0 if the expression is even.

Syntax

Even(x)

Odd(x)

x:aexp

Example

```
OpenW # 1
Local x = 6
Print "The value of x is "; x; "which is " &
  (Even(x) ? "even." : "odd.")
x = 3
Print "The value of x is "; x; "which is " &
  (Odd(x) ? "odd." : "even.")
```

See Also

Odd()

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

Inc, Incr Command

Purpose

Increments a numeric variable.

Syntax

Inc v

Incr v [, n = 1]

v:numeric variable
n:numeric exp

Description

Inc v increments the variable v by 1.

Incr v, n increments the variable v by n (default 1).

Example

```
OpenW # 1
Local x = 2.7
Inc x
Print x    // Prints 3.7
Incr x, 2.5
Print x    // Prints 6.2
```

Remarks

Although **Inc** can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed. Alternatives to **Inc** are:

```
x = x + 1
x := x + 1
x += 1
x++
Sub x, -1
Add x, 1
```

When integer variables are used **Inc** doesn't test for overflow!

See Also

Add, Sub, Dec, Succ, Pred, ++, --, +=, -=

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Dec Command

Purpose

Decrements a numeric variable.

Syntax

Dec x

Description

Dec x decrements the value of x by 1.

Example

```
OpenW # 1
Local x = 2.7
Dec x
Print x  // Prints 1.7
```

Remarks

Although **Dec** can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimisation for speed.

Instead of **Dec**

```
x = x - 1
x := x - 1
x -= 1
x--
Sub x, 1
```

```
Add x, -1
```

can be used instead.

When integer variables are used **Dec** doesn't test for overflow!

See Also

Inc, Add, Sub, Mul, Div, ++, --, +=, -=, /= , *=

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

Decr Command

Purpose

decrements a numeric variable

Syntax

Decr x [, y]

x:avar
y:aexp

Description

With the command **Decr** you decrements the vale of the variable x by 1 or by the given value of y.

Example

```
OpenW 1
Local a%
a = 10000
Decr a
Print a // Prints 9999
Decr a, 1500
Print a // Prints 8499
```

See Also

Add, Inc, Incr

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

Pred Function

Purpose

Calculates the first whole number smaller than an integer expression.

Syntax

x = **Pred**(n)

$ = **Pred**[**$**](a$)

x, n:integer expression

Description

Pred(n) returns the first whole number smaller than the integer expression n.

Pred(a$) returns a character whose ASCII value is one less than the first character of a string expression.

Example

```
OpenW # 1
Print Pred(4 * 11 - 1)     // Prints 42
Local l% = Pred(4 * 11 - 1)
Print l%                   // Prints 42
Print Pred("Hello World") // Prints G
```

Remarks

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

See Also

String, Add(), Sub(), Mul(), Div(), Mod(), Succ()

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

Succ Function

Purpose

Calculates the first natural number greater than an integer expression.

Returns a character whose ASCII value is one greater than the first character of a string expression.

Syntax

% = **Succ**(n)

$ = **Succ**[$](a$)

Description

Succ(n) returns the first natural number greater than the integer expression n.

Example

```
Debug.Show
Trace Succ(4 * 10 + 1)          // Prints 42
Trace Succ("Hello world")       // Prints I
```

Remarks

Succ(a$) corresponds to **Chr**$(**Succ**(**Asc**(a$))).

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

See Also

String, Pred()

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Mat Add Command

Purpose

Adds all elements in two (one- or two-dimensional) floating point arrays.

Syntax

Mat Add a() = b() + c() or

Mat Add a(), b() or

Mat Add a(), x

a(), b(), c():names of one- or two-dimensional floating point (Double) arrays

x:aexp

Description

Mat Add a()= b() + c() is only valid for floating point arrays of the same order, such as **Dim** a(n, m),b(n, m),c(n, m) or **Dim** a(n),b(n),c(n). The contents of elements in array c() are added to the contents of elements in array b() and the result is written to array a().

Mat Add a(), b() adds the contents of elements in array b() to the elements in array a() and writes the result to array a(). The original array a() is thereby lost.

Mat Add a(), x adds the expression x to the contents of all elements in array a() and writes the result to array a(). The original array a() is thereby lost.

Example

```
OpenW # 1 : FontName = "courier new"
Global Double a(1 .. 3, 1 .. 5)
Global Double b(1 .. 3, 1 .. 5)
Global Double c(1 .. 3, 1 .. 5)
Local x%
Mat Set b() = 3
Mat Set c() = 4
Mat Print b()
Print String$(9, "-")
Mat Print c()
Print String$(9, "-")
Mat Add a() = b()+c()
Mat Print a()
Erase a(), b(), c()
```

...or...

```
OpenW 1 : FontName = "courier new"
Global Double a(1 .. 3, 1 .. 5)
Global b#(1 .. 3, 1 .. 5), x%
Mat Set a() = 1
Mat Set b() = 3
Mat Print a()
Print String$(10, "-")
Mat Print b()
Print String$(10, "-")
Mat Add a(), b()
Mat Print a()
Erase a(), b()
```

...or...

```
OpenW 1 : FontName = "courier new"
Global Double a(1 .. 3, 1 .. 5)
Mat Set a() = 1
Mat Print a()
Print String$(10, "-")
Mat Add a(), 5
Mat Print a()
Erase a()
```

Remarks

Use the format of dimensioning Dim v#(1..n, 1..m) so the indexing doesn't depend on the **Option Base** setting.

Mat Base is no longer supported.

See Also

Mat Sub, Mat Mul

{Created by Sjouke Hamstra; Last updated: 13/10/2014 by James Gaite}

Mat Sub Command

Purpose

Subtracts all elements in two (one- or two-dimensional) floating point arrays.

Syntax

Mat Sub a()=b()-c()or

Mat Sub a(),b() or

Mat Sub a(),x

a(), b(), c():names of one- or two-dimensional floating point arrays
x:aexp

Description

Mat Sub a()=b()-c() is only valid for floating point arrays of the same order, such as **Dim** a(n, m),b(n, m),c(n, m) or **Dim** a(n),b(n),c(n). The contents of elements in array c() are subtracted from the contents of elements in array b() and the result is written to array a().

Mat Sub a(),b() subtracts the contents of elements in array b() from the elements in array a() and writes the result to array a(). The original array a() is thereby lost.

Mat Sub a(),x subtracts the expression x from the contents of all elements in array a() and writes the result to array a(). The original array a() is thereby lost.

Example

```
OpenW 1 : Win_1.FontName = "terminal"
Global Double a(1 To 3, 1 To 5)
Global Double b(1 To 3, 1 To 5)
Global Double c(1 To 3, 1 To 5)
Mat Set b() = 3
Mat Set c() = 4
Mat Print b(), 2, 0
divide(14, "Minus")
Mat Print c(), 2, 0
divide(14, "Equals")
Mat Sub a() = b()-c()
Mat Print a(), 2, 0
Print : Print : Print
Erase a(), b(), c()
Dim a(3, 5), b(3, 5), c(3, 5)
Mat Set b() = 3
Mat Set c() = 4
Mat Print b(), 2, 0
divide(17, "Minus")
Mat Print c(), 2, 0
divide(17, "Equals")
Mat Sub a() = b()-c()
Mat Print a(), 2, 0
Erase a(), b(), c()

Sub divide(n%, n$)
  Print
  Print String(n%, "-"); : Trace Win_1.CurrentX
  Text (CurrentX - TextWidth(n$ & "  ")) / 2,
    CurrentY, " " & n$ & " "
  Print : Print
EndSub
```

Example 2

```
OpenW 1 : Win_1.FontName = "terminal"
Global Double a(1 To 3, 1 To 5), x%
Global Double b(1 To 3, 1 To 5)
Mat Set a() = 1
Mat Set b() = 3
Mat Print a(), 2, 0
divide(14, "Minus")
Mat Print b(), 2, 0
divide(14, "Equals")
Mat Sub a(), b()
Mat Print a()
Erase a(), b()

Sub divide(n%, n$)
  Print
  Print String(n%, "-"); : Trace Win_1.CurrentX
  Text (CurrentX - TextWidth(n$ & "  ")) / 2,
   CurrentY, " " & n$ & " "
  Print : Print
EndSub
```

Example 3:

```
OpenW 1
Win_1.FontName = "Terminal"
Global Double a(1 To 3, 1 To 5), x%
Mat Set a() = 1
Mat Print a(), 2, 0
Print String$(14, "-")
Mat Sub a(), 5
Mat Print a()
```

Remark

-

See Also

Mat Add, Mat Mul

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Mat Mul Command

Purpose

Multiplies one- or two-dimensional floating point arrays which are interpreted as matrices.

Syntax

Mat Mul a()=b()*c() or

Mat Mul x=a()*b() or

Mat Mul a(),x

a(),b(),c():names of one- or two-dimensional floating point arrays

x:aexp

Description

Mat Mul a()=b()*c() is intended for 'related' matrices of the same order. Matrices b() and c() are multiplied. The result of this multiplication is written to matrix a(). In order to get a product of a matrix multiplication, the matrix on the left (in this case matrix b()) must have the same number of columns as the matrix on the right (in this case c()) has rows.

The matrix a() must, in this example, have the same number of rows as b() and the same number of columns as c(), i.e. Dim a(2,2),b(2,3),c(3,2)

Matrices are multiplied using the formula 'rows times columns'. I.e. the elements in a(i,j) are obtained by multiplying the elements of the i-th row in matrix b() with the j-th column in matrix c() and the individual products are added up. If vectors are used instead of matrices, **Mat Mul** a()=b()*c() produces the dyadic product of two vectors.

Mat Mul x=a()*b() is intended for vectors with the same number of elements. The result x is the scalar product of vectors a() and b(). The scalar product of two vectors is defined as the sum of n products a(i)*b(i), i=1,...,n.

Mat Mul a(),x multiplies the matrix or vector a() with the expression x.

Example

```
OpenW # 1
Global Double a(1 .. 2, 1 .. 2)
Global Double b(1 .. 2, 1 .. 3)
Global Double c(1 .. 3, 1 .. 2)
Mat Set b() = 1
Data 1,2,-3,4,5,-1
Mat Read c()
Mat Print b(), 5, 1
Print String$(18, "-")
Mat Print c(), 5, 1
Print String$(18, "-")
Mat Mul a() = b()*c()
Mat Print a(), 5, 1
Erase a(), b(), c()
```

...and...

```
Global Double a(1 .. 3, 1 .. 3
Global Double b(1 .. 3), c(1 .. 3)
Data 1,2,-3,4,5,-1
```

```
Mat Read b()
Mat Read c()
Mat Print b(), 5, 1
Print String$(18, "-")
Mat Print c(), 5, 1
Print String$(18, "-")
Mat Mul a() = b()*c()
Mat Print a(), 5, 1
Erase a(), b(), c()
```

...and...

```
OpenW 1 // Mat Mul x = a()*b()
Global Double b(1 .. 3), c(1 .. 3), x%
Data 1,2,-3,4,5,-1
Mat Read b()
Mat Read c()
Mat Print b(), 5, 1
Print String$(18, "-")
Mat Print c(), 5, 1
Print String$(18, "-")
Mat Mul x = b()*c()
Print x
Erase b(), c()
```

Remarks

-

See Also

Mat Add, Mat Sub

{Created by Sjouke Hamstra; Last updated: 14/10/2014 by James Gaite}

Mat Cpy Command

Purpose

copies a number of rows with a number of elements, from row/column offset in the source matrix to row/column offset in the target matrix.

Syntax

Mat Cpy a([i, j])=b([k, l])[,h, w]

i, j, k, l, w, h:integer expression

a(), b():one or two dimensional floating point arrays

Description

Mat Cpy a([i, j])=b([k, l])[,h, w] copies h rows with w elements in matrix b(), from l and k row/column offset in matrix b() to i and j row/column offset in matrix a(). The maximum number of elements copied is equivalent to the minimum number allowed when dimensioning the matrices, the number of rows (h) and the number of elements per row (w).

If **Mat Cpy** is used on vectors j and l are ignored. Following a **Dim** a(n),b(m) the a() and b() are interpreted as row vectors, that is to say as matrices of type (1,n) and (1,m).

To handle a() and b() as column vectors, they must be dimension as matrices of type (n,1) and (m,1), that is to say as Dim a(n,1),b(m,1).

Mat Cpy always handles vectors as column vectors, regardless of their type, so in order to use the correct **Mat Cpy** syntax with vectors **Mat Cpy** a(n,1)=b(m,1) must always be used.

If the h and w parameters in **Mat Cpy** are given explicitly, the following rules apply when copying vectors:

When w => 1 only the h parameter is taken into account. When w=0 no copying takes place.

When h =>1, the w is taken into account only when b() is a row vector and a() is a column vector. Here too, no copying takes place when h=0.

Example

```
OpenW 1
Global Double a(1 .. 3, 1 .. 5)
Global Double b(1 .. 6, 1 .. 6)
Mat Set a() = 1
Mat Set b() = 5
Mat Cpy a(2, 2) = b(3, 4), 3, 3
Mat Print a()
```

Prints:

```
1,1,1,1,1
1,5,5,5,1
1,5,5,5,1
```

Remarks

If some indices are dropped - due to the given width (w) or height (h) - **Mat Cpy** can result in the following special cases:

Mat Cpy a() = b()

copies into matrix a() all elements of matrix b() for which there are identical indices in matrix a() as in the following example:

```
OpenW 1
Global Double a(1 .. 3, 1 .. 5)
Global Double b(1 .. 6, 1 .. 6)
Mat Set b() = 5
Mat Cpy a() = b()
Mat Print a()
```

prints

```
5,5,5,5,5
5,5,5,5,5
5,5,5,5,5
```

Mat Cpy a(i, j)=b()

copies all elements in matrix b(), from row/column offset defined with Mat BASE, to row/column offset defined with i and j in matrix a(). The maximum number of elements copied is equivalent to the minimum number allowed when dimensioning the matrices, the number of rows (h) and the number of elements per row (w). Example:

```
OpenW 1
Global Double a(1 .. 3, 1 .. 5)
Global Double b(1 .. 6, 1 .. 6)
Mat Set a() = 1
Mat Set b() = 5
Mat Cpy a(2, 2) = b()
Mat Print a()
```

Prints:

```
1,1,1,1,1
1,5,5,5,5
1,5,5,5,5
```

Mat Cpy a() = b(k, l)

copies all elements in matrix b(), from row/column offset defined with k and l, to row/column offset defined with Mat BASE in matrix a(). The maximum number of elements copied is equivalent to the minimum number allowed when dimensioning the matrices, the number of rows (h) and the number of elements per row (w). Example:

```
OpenW 1
Global Double a(1 .. 3, 1 .. 5)
Global Double b(1 .. 6, 1 .. 6)
Mat Set a() = 1
Mat Set b() = 5
Mat Cpy a() = b(4, 4)
Mat Print a()
```

Prints:

```
5,5,5,1,1
5,5,5,1,1
5,5,5,1,1
```

Mat Cpy a(i, j) = b(k, l)

copies all elements in matrix b(), from row/column offset defined with k and l, to row/column offset defined with i and j in matrix a(). The maximum number of elements copied is equivalent to the minimum number allowed when dimensioning the matrices, the number of rows (h) and the number of elements per row (w). Example:

```
OpenW # 1
Global Double a(1 .. 3, 1 .. 5)
Global Double b(1 .. 6, 1 .. 6)
Mat Set a() = 1
Mat Set b() = 5
Mat Cpy a(2, 2) = b(4, 4)
Mat Print a()
```

Prints:

```
1,1,1,1,1
1,5,5,5,1
1,5,5,5,1
```

Mat Cpy a()=b(), h, w

copies h rows and w elements in matrix b(), from row/column offset defined with Mat BASE, to row/column offset matrix a(). The maximum number of elements copied is equivalent to the minimum number allowed when dimensioning the matrices, the number of rows (h) and the number of elements per row (w). Example:

```
OpenW # 1
Global Double a(1 .. 3, 1 .. 5)
Global Double b(1 .. 6, 1 .. 6)
Mat Set a() = 1
Mat Set b() = 5
Mat Cpy a() = b(), 3, 3
Mat Print a()
```

Prints:

```
5,5,5,1,1
5,5,5,1,1
5,5,5,1,1
```

See Also

MatX Cpy, Mat Trans

{Created by Sjouke Hamstra; Last updated: 14/10/2014 by James Gaite}

Mat XCpy Command

Purpose

Copies a specified number of rows containing a specified number of elements, from the given row/column offset in source matrix to the given row/column offset in target matrix. The source matrix, or the relevant part of it, are internally transposed before copying.

Syntax

Mat XCpy a([i, j])=b([k, l])[,h, w]

i, j, k, l, w, h:integer expression

a(),b():one- or two-dimensional floating point array

Description

Mat XCpy a([i, j])=b([k, l])[,h, w] copies h rows with w elements, from row/column offset defined with l and k in matrix b(), to row/column offset defined with i and j in matrix a(). The maximum number of elements copied is equivalent to the minimum number allowed when dimensioning the matrices, the number of rows (h) and the number of elements per row (w). The matrix b(), or the relevant part of it, are internally transposed before copying, that is to say the rows and column are swapped. This change affects only the copy and not the matrix b() itself.

If **Mat XCpy** is used on vectors j and l are ignored. Following a **Dim** a(n),b(m) the a() and b() are interpreted as row vectors, that is to say as matrices of type (1,n) and (1,m).

To handle a() and b() as column vectors, they must be dimension as matrices of type (n,1) and (m,1), that is to say as Dim a(n,1),b(m,1).

If both vectors are of the same type, that is to say they are both rows or columns, **Mat Cpy** must be used.

If the h and w parameters in **Mat XCpy** are given explicitly, the following rules apply when copying vectors:

When w => 1, the h parameter is taken into account only when b() is a column vector and a() is a row vector. When w=0 no copying takes place.

When h => 1 the w parameter is taken into account only when b() is a row vector and a() is a column vector. When h=0 no copying takes place.

Example

```
OpenW # 1
Global Double a(1 To 3, 1 To 5), x%
Global Double b(1 To 7, 1 To 2)
Mat Set a() = -1
Mat Set b() = 5
Mat Print a(), 2, 0
Print
Mat Print b(), 2, 0
Print
Mat XCpy a(1, 2) = b(3, 2)
Mat Print a(), 2, 0
```

Remarks

If some indices are dropped - due to the given width (w) or height (h) - the following special cases can result just like with **Mat Cpy**:

Mat XCpy a()=b()

Mat XCpy a([i,j])=b()

Mat XCpy a()=b([k, l])

Mat XCpy a()=b(),w, h

These act the same as the corresponding **Mat Cpy** commands, except for the transposition of relevant areas of matrix b() before copying to matrix a(). The b() matrix remains unchanged!

See Also

Mat Cpy, Mat Trans

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Mat Trans Command

Purpose

Copies a transposed source matrix into a target matrix.

Syntax

Mat Trans a()=b()

a(),b():one- or two-dimensional floating point array

Description

Mat Trans a()=b() copies the transposed matrix b() into matrix a(), assuming that both a() and b() are dimensioned appropriately, that is to say the number of rows in a() must correspond to the number of columns in b(), and the number of columns in a() must correspond to the number of rows in b() (for example **Dim** a(n,m),b(m,n)).

Example

```
Global Double a(1 To 4, 1 To 3)
Global Double b(1 To 3, 1 To 4)
Mat Set a() = 2
Mat Set b() = 5
Mat Print a()
Print
Mat Print b()
Print
Mat Trans a() = b()
Mat Print a()
```

Remarks

Defines a square matrix, that is to say, a matrix with the same number of rows and columns so that **Mat Trans** a() can be used. This command swaps the rows and columns in matrix a() and writes the modified matrix back to a(). The original matrix a() is thereby lost. (However, it can be restored by performing **Mat Trans** a() again.)

See Also

Mat Cpy, Mat XCpy

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Mat Clr Command

Purpose

Sets all elements in a one- or two-dimensional floating point array to 0.

Syntax

Mat Clr a()

a():name of a one- or two-dimensional floating point (Double) array

Description

Mat Clr a() is equivalent to **ArrayFill** a(),0, that is to say the command sets all elements of array a() to 0.

Example

```
OpenW 1
Global Double a(1 .. 3, 1 .. 3)
Data 1,2,3,4,5,6,7,8,9
Mat Read a()
Mat Print a()
Print "---------"
Mat Clr a()
Mat Print a()
```

First it prints 1 to 9, and then all 0s.

See Also

ArrayFill, Mat Set, Mat One, Mat Neg

{Created by Sjouke Hamstra; Last updated: 14/10/2014 by James Gaite}

MatSet Command

Purpose

Assigns a value to all elements of a one- or two-dimensional floating point array.

Syntax

Mat Set a()=x

a():name of a one- or two-dimensional floating point array

x:aexp

Description

Mat Set a()=x is equivalent to an **ArrayFill** a(),x, i.e. the command sets all elements of the array a() to value x.

Example

```
OpenW # 1
PrintScroll = True
Global Double a(1 To 5, 1 To 7), i%, j%, x%
For i% = 1 To 5
  For j% = 1 To 7
    a(i%, j%) = Rand(10)
  Next j%
Next i%
Mat Set a() = 5.3
For i% = 1 To 5
  For j% = 1 To 7
    Print a(i%, j%)
  Next j%
Next i%//prints the value 5.3 35 times
```

See Also

ArrayFill, Mat Clr, Mat One, Mat Neg

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Mat One Command

Purpose

Creates a unitary matrix.

Syntax

Mat One a()

a():name of a two-dimensional floating point array with the same numberof rows and columns

Description

Mat One a() creates, from a two dimensional floating point array a() with the same number of rows and columns, an array in which the elements a(1,1), a(2,2), ...,a(n,n) are equal to 1 and all other elements are equal to 0.

Example

```
OpenW # 1
Global Double a(1 ... 3, 1 ... 3)
Mat One a()
Mat Print a()
```

prints:

```
1,0,0
0,1,0
0,0,1
```

See Also

ArrayFill, Mat Clr, Mat Set, Mat Neg

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Mat Neg Command

Purpose

Negates all elements in a one- or two-dimensional floating point array

Syntax

Mat Neg a()

a():name of a one- or two-dimensional floating point array

Description

Mat Neg a() multiplies all elements of a one or two dimensional floating point array a() with -1.

Example

```
OpenW 1
Global Double a(1 .. 3, 1 .. 3)
Mat One a()
Mat Print a()
Print
Mat Neg a()
Mat Print a()
```

Remarks

-

See Also

Mat Clr, Mat Set, Mat One

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Mat Det Command

Purpose

Calculates the determinant of a two-dimensional floating point array which is interpreted as a matrix.

Syntax

Mat Det x=a([i, j])[,n]

a():name of a two-dimensional floating point array

x:aexp

i, j, n:integer expression

Description

Mat Det x=a([i, j])[,n] calculates the determinant of a square matrix of type (n,n). A determinant of a square section of a matrix can also be calculated. This matrix section is defined by i and j for row and column offsets in a() and by n for the number of elements. An internal matrix of (n,n) type is thereby created at i-th row and j-th column.

Example

```
OpenW # 1
Data 2,4.5,6,3.2,7,1.7,-4,12
Data -3,5,9,-2.1,6,9,11,3
Data 11.4,2.3,6,3.2,6,1.2,-5,7
Data 3,5,6,8.2,4.1,-5.2,6.2,7.9
Data 1,2.3,9,8.1,0,4.2,5,3.7
Data 4.2,7.1,8.3,9.1,-5,-3,-1,0
Data 2.0,3,9.1,0,0,7.1,-3,8.8
Data 2.1,9,3.3,4,5,-1,-2,0
Global Double a(1 .. 8, 1 .. 8), x, y, z
Global Double b(1 .. 4, 1 .. 4), k%
Mat Read a()
Mat Print a(), 5, 2 // original matrix
Print
//to calculate the determinant
Mat Det x = a()
Print "Determinant = "; x
Print
Print "Press any Key"
KeyGet k%
Cls
Mat Det y = a(3, 2), 4//calculates the determinant
//of a matrix segment
Print "Segment determinant= "; y
Print
Mat Cpy b() = a(3, 2), 4, 4
Mat Print b(), 5, 2
Print
Mat Det z = b()
Print "Determinant = "; z
```

See Also

Mat QDet, Mat Rank, Mat Inv

{Created by Sjouke Hamstra; Last updated: 14/10/2014 by James Gaite}

Mat QDet Command

Purpose

calculates the determinant of a two-dimensional floating point array which is interpreted as a matrix.

Syntax

Mat QDet x=a([i, j])[,n]

a():name of a two dimensional floating point array

x:aexp

i, j, n:integer expression

Description

Mat QDet x=a([i, j])[,n] is equivalent to **Mat Det** x = a([i, j])[,n] except that it's optimized for speed not accuracy. As a rule both methods deliver the same result. However, **Mat Det** should always be used in case of 'critical' matrices whose determinant is close to 0.

Example

```
OpenW # 1
Data 2,4.5,6,3.2,7,1.7,-4,12
Data -3,5,9,-2.1,6,9,11,3
Data 11.4,2.3,6,3.2,6,1.2,-5,7
Data 3,5,6,8.2,4.1,-5.2,6.2,7.9
Data 1,2.3,9,8.1,0,4.2,5,3.7
Data 4.2,7.1,8.3,9.1,-5,-3,-1,0
Data 2.0,3,9.1,0,0,7.1,-3,8.8
Data 2.1,9,3.3,4,5,-1,-2,0
Global Double a(1 To 8, 1 To 8), x, y, k%
Mat Read a()
Mat Print a(), 4, 1
Print
Mat Det x = a()//calculate the determinant
Print "Determinant with Mat Det = "; x
Print
Mat QDet y = a()//calculate the determinant
Print "Determinant with Mat QDet = "; y
Print
Print "Deviation = "; x - y
End
```

See Also

Mat Det, Mat Rank, Mat Inv

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Mat Rank Command

Purpose

Returns the rank of a two-dimensional floating point array which is interpreted as a matrix.

Syntax

Mat Rank x=a([i, j])[,n]

Mat Rang x=a([i, j])[,n]

a():name of a two-dimensional floating point array
x:aexp
i, j, n:integer expression

Description

Mat Rank x=a([i, j])[,n] prints the rank of a square matrix. Analogous to **Mat Det** and **Mat QDet** an arbitrary row and column offset can be specified.

To process a section of a matrix, a number of elements is specified in n. An internal matrix of (n, n) type is thereby created at row i and column j.

Example

```
OpenW # 1 : Win_1.FontName = "courier new"
Data 2,4.5,6,3.2,7,1.7,-4,12
Data -3,5,9,-2.1,6,9,11,3
Data 11.4,2.3,6,3.2,6,1.2,-5,7
Data 3,5,6,8.2,4.1,-5.2,6.2,7.9
Data 1,2.3,9,8.1,0,4.2,5,3.7
Data 4.2,7.1,8.3,9.1,-5,-3,-1,0
Data 2.0,3,9.1,0,0,7.1,-3,8.8
Data 2.1,9,3.3,4,5,-1,-2,0
Global Double a(1 ... 8, 1 ... 8), x
Mat Read a()
Mat Print a(), 2, 0
Print
Mat Rank x = a()//calculate the rank
Print "Rank = "; x
```

See Also

Mat Det, Mat QDet, Mat Inv

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Mat Inv Command

Purpose

Calculates an inverse of a two-dimensional floating point array which is interpreted as a matrix.

Syntax

Mat Inv a()=b()

a(),b():names of two-dimensional floating point arrays with the same number of rows and columns.

Description

Mat Inv a()=b() returns the inverse of a square matrix. The inverse of matrix b() is written to matrix a(). a() must, therefore, be of the same type as b().

Example

```
Data 2,4.5,6,3.2,7,1.7,-4,12
Data -3,5,9,-2.1,6,9,11,3
Data 11.4,2.3,6,3.2,6,1.2,-5,7
Data 3,5,6,8.2,4.1,-5.2,6.2,7.9
Data 1,2.3,9,8.1,0,4.2,5,3.7
Data 4.2,7.1,8.3,9.1,-5,-3,-1,0
Data 2.0,3,9.1,0,0,7.1,-3,8.8
Data 2.1,9,3.3,4,5,-1,-2,0
OpenW # 1
Global Double a(1 .. 8, 1 .. 8)
Global Double b(1 .. 8, 1 .. 8)
Global Double c(1 .. 8, 1 .. 8)
Global Double d(1 .. 8, 1 .. 8), a%
Mat Read b()
Mat Print b(), 6, 3
Print
Print "Inverse:"
Print
Mat Inv a() = b() //calculate the inverse
Mat Print a(), 6, 3
Print
Print "Press any key"
KeyGet a%
Cls
Print "Original matrix * Inverse "
Print
Mat Mul d() = b() * a()
Mat Print d(), 6, 3
```

See Also

Mat Det, Mat QDet, Mat Rank

{Created by Sjouke Hamstra; Last updated: 14/10/2014 by James Gaite}

Mat Print Command

Purpose

Prints the elements of an array to screen or a channel.

Syntax

Mat Print [#i,]a()[, g, n]

a():name of a floating point array
i, g, n:integer expression

Description

Mat Print [#i,]a()[,g,n] prints a floating point array to screen. One-dimensional floating point arrays are printed on one line with individual elements separated by commas. For two-dimensional arrays a line feed is performed after each row. Similar to the Print command, the output can optionally be redirected with #i. g and n cause the formatting of the numbers similar to **Str**$(x,g,n).

Example

```
OpenW # 1
Data 1,2.33333,3
Data 7,5.25873,9.376
Data 3.23,7.2,8.999
Global Double a(1 To 3, 1 To 3)
Mat Read a()
Mat Print a()
Print
Mat Print a(), 5, 3
```

```
Print
Mat Print a(), 6, 3
```

See Also

Mat Read

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Mat Read Command

Purpose

Reads values from Data lines into a floating point array.

Syntax

Mat Read a()

a():name of a floating point array

Description

-

Example

```
Option Base 1
OpenW # 1
Data 1,2,3,4,5,6,7,8,9,10
Dim a(2, 5) As Double
Mat Read a()
Mat Print a(), 2, 0
Print
Print a(2, 4) //prints 9
```

See Also

Mat Print

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Mat Norm Command

Purpose

Row- or column-wise normalizing of a two-dimensional floating point array which is interpreted as a matrix.

Syntax

Mat Norm a(),i

a():name of a two-dimensional floating point array

i:ivar; i=0 for row-wise and i=1 for column-wise normalizing

Description

Mat Norm a(),0 and **Mat Norm** a(),1 are used for both matrices and vectors. **Mat Norm** a(),0 normalizes a matrix (or a vector) row-wise and **Mat Norm** a(),1 normalizes a matrix (or a vector) column-wise. This means that in case of row-wise (column-wise) normalizing the sum of squares of all elements in each row (column) is equal to 1.

Example

```
OpenW 1
Global a%, n% = 8, k%, i%
Global Double a(1 To n%, 1 To n%)
Global Double b(1 To n%, 1 To n%)
Global Double v(1 To n%), v(), x
Data 1,2,3,4,5,6,7,8
Data 3.2,4,-5,2.4,5.1,6.2,7.2,8.1
Data -2,-5,-6,-1.2,-1.5,-6.7,4.5,8.1
Data 5,-2.3,4,5.6,12.2,18.2,14.1,16
Data 4.1,5.2,16.7,18.4,19.1,20.2,13.6,14.8
Data 15.2,-1.8,13.6,-4.9,5.4,19.8,16.4,-20.9
Data -3.6,6,-8.2,-9.1,4,-2.5,2,3.4
Data 4.7,8.3,9.4,10.5,11,19,15.4,18.9
//
Mat Read a()
//save the original matrix
Mat Cpy b() = a()
Print "Original Matrix"
Print
Mat Print a(), 7, 2
KeyPress
//
// row-wise normalising
//
Mat Norm a(), 0
Print "Row-wise normalised: "
Print
Mat Print a(), 7, 2
KeyPress
//
// testing of the row-wise normalising
//
Print "Test: "
Print
For i% = 1 To n%
  Mat XCpy v() = a(i%, 1) // copies a() row-wise
    into vector v()
  Mat Mul x = v()*v() // calculates the scalar
    product of v() and v()
  Print x`
Next i%
KeyPress
// column-wise normalising
Mat Cpy a() = b()//copy the original matrix
Mat Norm a(), 1
Print "Column-wise normalised: "
Print
Mat Print a(), 7, 2
KeyPress
// testing of column-wise normalising
Print "Probe : "
Print
For i% = 1 To n%
  Mat Cpy v() = a(1, i%) // copies a() column-wise
    into vector v()
  Mat Mul x = v()*v()// calculates the scalar
    product of v() and v()
  Print x`
Next i%
KeyPress
CloseW 1

Sub KeyPress
  Local a%
  Print
  Print "Press any key"
  KeyGet a%
  Cls
EndSub
```

See Also

-

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

}

cAlloc Function

Purpose

Allocates an array in memory with elements initialized to 0.

Syntax

long = **cAlloc**(*num*, *size*)

num, size:iexp

Description

cAlloc() returns a pointer to the allocated space. *num* specifies the number of elements and *size* specifies the length in bytes of each element. The reserved memory block is initialized with 0.

cAlloc() is implemented to easily port C-source code. Compare the internal implementation in both C and GFA-BASIC 32:

The allocated memory can be resized using **mReAlloc** or **mShrink** and released with **MFree**.

The C- implementation

```
void *calloc(int a, int b)
{
    void *p = malloc(a * b);
    if(p) memset(p, 0, a * b);
    return p;
}
```

The GFA-BASIC 32 implementation

```
Function cAlloc(a As Int, b As Int) As Int
  Local p As Int = mAlloc(a * b)
  If(p) Then MemSet(p, 0, a * b)
  Return p
End Func
```

Example

```
Dim p As Long = cAlloc(10, SizeOf(Int))
```

Allocates 40 bytes (10 * 4), because the size of an Int data type is 4 bytes.

Remarks

C	**GFA-BASIC 32**
malloc	**mAlloc**
calloc	**cAlloc**
realloc	**mReAlloc** or **mShrink**
free	**mFree**
memset(a, v, n)	**MemSet**(a, v, n) or **MemBFill** a, n, v
memcpy(d, s, n)	**MemCpy**(d, s, n)

See Also

mAlloc(), mFree(), mShrink(), mReAlloc()

{Created by Sjouke Hamstra; Last updated: 03/03/2017 by James Gaite}

Memory Allocation

Much of the memory allocation required within a program is handled by GFABasic's commands and functions. However, every now and again, an occasion will arise when having direct access to reserved memory is preferable or the only way to carry out a task, and for that reason the following commands and their Window API equivalents have been included in GFABasic's list of commands and functions.

Using GFABasic Show

Using Windows APIs Show

Remarks & Comparisons Show

See Also

cAlloc

{Created by James Gaite; Last updated: 06/03/2017 by James Gaite}

Bmove and BlockMove Commands

Purpose

Copies an area of memory.

Syntax

BMove from%, to%, count%
BlockMove from%, to%, count%

from%, to%:address
count:integer expression

Description

BMove and **BlockMove** are synonymous and are used to copy memory areas. The copy is performed from address from% to the address to%. The number of bytes to copy is specified in count%.

Example

```
OpenW # 1
Local i%, j%
Local Double a(3, 3), b(3, 3)
For i% = 0 To 3
  For j% = 0 To 3
    a(i%, j%) = Random(2000 - 1000)
  Next j%
Next i%
Print "BEFORE:"
```

```
Print
Print "Array a()"
Print
Mat Print a()
Print "------------------"
Print "Array b()"
Print
Mat Print b()
Print
BMove V:a(0, 0), V:b(0, 0), Dim?(a()) * 8
Print "AFTER BMove:"
Print
Print "Array a()"
Print
Mat Print a()
Print "------------------"
Print "Array b()"
Print
Mat Print b()
```

First, two arrays are dimensioned. Array a() is then filled with random numbers. **V:** a(0,0) returns the address of the first element in a(), **V:** b(0,0) the first element in b(). Each floating point variable requires eight bytes of memory. The number of elements in a() is deter-mined with Dim?(a()). Dim?(a())*8 returns then the number of bytes to be copied.

Remarks

The copying of array a() into array b() in the above example can also be done with

```
For i% = 0 To 3
  For j% = 0 To 3
    b(i%, j%) = a(i%, j%)
  Next j%
Next i%
```

The **BMove** and **BlockMove** commands, however, requires less memory and are - depending on the contents being copied - up to 100 times faster.

See Also

MemCpy

{Created by Sjouke Hamstra; Last updated: 11/01/2017 by James Gaite}

MemCpy

Purpose

Copies a block of memory in fastest possible way.

Syntax

MemCpy dst, src, cnt

MemCpy(dst, src, cnt)

Description

The first parameter of **MemCpy** is the address of the destination and the second one the one of the source and the third one can be a constant or, for example, the length of the source to copy.

MemCpy is extremely efficient in copying Type variables. **MemCpy** is one of the rare commands that is compiled inline when *cnt* is a constant (not a function).

Example

```
Local a$ = "GFA Basic", b$ = Space(9)
MemCpy V:b$, V:a$, 9                // This works as
 described
Print a$, b$
a$ = "GFA Basic", b$ = Space(9)
MemCpy V:b$, V:a$, Len(b$)          // This doesn't
 work this way...
Print a$, b$
a$ = "GFA Basic", b$ = Space(9)
```

```
MemCpy V:a$, V:b$, Len(b$)        // ..but for some
  reason, does this way
Print a$, b$
a$ = "GFA Basic", b$ = Space(9)
MemCpy V:b$, V:a$, 9              // Once again,
  this one works fine
Print a$, b$
```

Remarks

MemCpy is highly compatible to the C function *memcpy*(). If the source and destination overlap, this function does not ensure that the original source bytes in the overlapping region are copied before being overwritten. Use **MemMove**, **Bmove**, or **BlockMove** to handle overlapping regions.

See Also

BMove, BlockMove, MemMove

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

Pause Command

Purpose

Interrupts a program.

Syntax

Pause n

n:integer expression

Description

Pause *n* interrupts a program for *n*/18.2 seconds.

Example

```
OpenW # 1 : AutoRedraw = 1
Print "Coffee break!"
Pause 182 //a ten second pause
Print "Coffee break is over"
```

See Also

Delay

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Delay Command

Purpose

interrupts a program for a number of seconds.

Syntax

Delay a

a:aexp

Description

Delay a interrupts a program for 'a' seconds.

Example

```
OpenW 1
Print "This window will stay open for 5 seconds
 only"
Delay 5
CloseW 1
```

Remarks

In contrast to **Pause** (dependent on the operating system) the time specified with **Delay** is portable. Delay uses the system clock.

See Also

Pause

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

FreeDll Command

Purpose

releases a DLL (dynamic link library)

Syntax

FreeDll filename$

Description

FreeDll explicitly releases a DLL from memory. The argument filename$ should be exactly the same as the DLL name specified in the **Declare** statement. Filename$ may contain a path.

Example

```
Declare FunctionA WNetAddConnection Lib "mpr.dll"
  (ByVal lpszNetPath As String, ByVal lpszPassword
 As String, ByVal lpszLocalName As String) As Long
// Use Dll
// release DLL
FreeDll "mpr.dll"
```

Remarks

When a DLL function is invoked after its DLL has been released, the DLL is reloaded. This due to the nature of **Declare**, which instructs the compiler to generate code to check for a valid DLL before calling a DLL function.

See Also

Declare

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Shell Command

Purpose

Invokes the command interpreter

Syntax

Shell t

x% = **Shell**(t)

t:sexp
x%iexp, return value

Description

Shell runs the command interpreter and so enables execution of DOS commands from within a GFA-BASIC 32 program.

Example

```
Shell "CHKDSK a: /f"      //tests the disk in
 drive A:
Shell "command.com"     // invokes Command.Com
 (Windows 9.x)
Shell "cmd"             // calls cmd NT, 2000, XP
Dim x% = Shell("Dir /4 | More")
Debug.Show
Debug.Print "Return value of Shell = ";x%
```

Remarks

With **Open** "CONOUT$" **For Output As** and *AllocConsole*() a command console can be opened and gives the application access to the input and output in the console. See Open.

See Also

Exec, ShellExec, System, WinExec

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

System Command

Purpose

Loads and runs a program.

Syntax

System "file$ [parameters]" [, options]

ret_large **= System**("file$ [parameters]")

file$:sexp

Description

The **System** command invokes the program *file$.* The *file$* expression contains the name of the called program. The program name includes the full pathname and the command line which is inserted in the program segment prefix of the called program.

The **System**(file$) function return a 64-bit integer, which is 0 in case of an error. Otherwise, the low order 32 bits contain the process handle of the program, and the high 32 bits the process ID. The first example shows how to start an external program and wait for it to end.

System is based on the API function *CreateProcess*, which can take quite some options. Many of these options are implemented in GFA-BASIC 32. There is an option to wait for a program to end, like: **System** "notepad", **Wait**. The return values of *CreateProcess* can be retrieved using the options as well. **System** "notepad", **ProcessID** pid% returns the process identification in pid%.

System supports the following options.

Dir "path"	specifies the current drive and directory for the child process.
App "programname"	Statement for the name of the program to start; mostly the usage of a command line will make more sense, but nevertheless, may be useful for someone.
Show SW_const	One of the SW_ constants. For GUI processes this specifies the default value the first time *ShowWindow* is called.
Pos x, y	Specifies the x and y offsets, in pixels, of the upper left corner of a window if a new window is created.
Size w, h	Specifies the width and height, in pixels, of the window if a new window is created.
Full	Full DOS-BOX
Fill attrib	Specifies the initial text and background colors if a new console window is created in a console application. This value can be any combination of the following values: FOREGROUND_BLUE, FOREGROUND_GREEN, FOREGROUND_RED, FOREGROUND_INTENSITY, BACKGROUND_BLUE, BACKGROUND_GREEN, BACKGROUND_RED, and BACKGROUND_INTENSITY.
Count cx, cy	For console processes, if a new console

	window is created, cx specifies the screen buffer width in character columns, and cy specifies the screen buffer height in character rows. These values are ignored in GUI processes.
Title sexp	For console processes, this is the title displayed in the title bar if a new console window is created.
Desktop sexp	string that specifies either the name of the desktop only or the name of both the desktop and window station for this process. A backslash in the string indicates that the string includes both desktop and window station names.
FeedOn	Use application starting cursor.
FeedOff	Don't use application starting cursor.
Wait	The command System waits in a loop with *MsgWaitForMultipleObjects*(..., 1000,...) until the end of the started process and executes, if necessary DoEvents in such a way that the program remains accessible.
ForceWait	Like **Wait**, but waiting cannot be interrupted. The calling program is actually disabled.
hProcess var	Copy the return value of the process handle into *var*% (the variable var is Long/Int or Handle).
hThread var	Copy the primary thread handle into *var* (Long/Int or Handle).
ProcessID var	Copy the process ID into *var* (Long/Int).
ThreadID var	Copy the thread ID into *var* (Long/Int).
ExitCode var	Copy the exit code into *var* (Long/Int). Without **Wait** or **ForceWait** mostly

	STATUS_PENDING (0x103)
StdIn h	Specifies a handle that will be used as the standard input handle to the process.
StdOut h	Specifies a handle that will be used as the standard output handle to the process
StdErr h	Specifies a handle that will be used as the standard error handle to the process
Inherit	Inherits handles from the calling process. Each inheritable open handle in the calling process is inherited by the new process. Inherited handles have the same value and access privileges as the original handles.
Advanced options	
Debug	The caller is a debugger, the new process is a process being debugged.
DebugThis	If not specified and the calling process is being debugged, the new process becomes another process being debugged by the calling process's debugger. If the calling process is not a process being debugged, no debugging-related actions occur.
Suspend	The called program is waiting for the *ResumeThread* (for debugger).
Detached	For console processes, the new process does not have access to the console of the parent process.
NewConsole	The new process has a new console, instead of inheriting the parent's

	console. This flag cannot be used with the **Detached** option.
Normal, Idle, High, RealTime	Controls the new process's priority class, which is used in determining the scheduling priorities of the process's threads. (Idle = background process, like a screen saver; High = the process will get 'all' processor time; RealTime = process can get all available processor time.
NewPGroup	The new process is the root process of a new process group.
Separate, Shared	Only valid when starting a 16-bit Windows-based application. The disadvantage of running **Separate** is that it takes significantly more memory to do so. You should use this flag only if the user requests that 16-bit applications should run in them own VDM. Shared overrides the system default setting and runs the new process in the shared Virtual DOS Machine.
DOS	Starts the program a real MSDOS application (not Win 95/98/Me).
DefErr	The default error mode is valid (not Win 95, 98, Me).
ProfUser, ProfKernel, ProfServer	The program is a user program, a kernel, or a server application and should use the appropriate profiles.

The use of **hProcess** and **hThread** require a *CloseHandle*. **ProcessID**, **ThreadId**, **hProcess,** and **hThread** aren't very useful together with **Wait** or **ForceWait.**

```
Dim id%, h%
```

```
System "notepad", ProcessID id%, hProcess h%
```

In case of an error (System returns 0) a message box is displayed.

Example

1 - Start notepad and wait.

```
OpenW Center 1
Local pHdl As Handle, pID As Int
Local l As Large, e%, h%
l = System("Notepad")
If !l Then Message _
  "Can't start Notepad" : End
pHdl = LoLarge(l) ' process handle
pID = HiLarge(l)  ' process ID
~GetExitCodeProcess(pHdl, V:e)
While e = STATUS_PENDING
  ~MsgWaitForMultipleObjects(1, V:pHdl, _
    0, 1000, QS_ALLINPUT)
  Beep -1
  DoEvents
  ~GetExitCodeProcess(pHdl, V:e)
Wend
~CloseHandle(pHdl)
```

Example 2

```
OpenW 1
Global a As Large, b$
b$ = " c:\test.dat"
If Exist(WinDir + "\notepad.exe")
  a = System(WinDir + "\notepad.exe" & b$)
  Message "Return value: " & Format(a)
Else
  Message "Program not found"
```

```
EndIf
Do
  Sleep
Until Me Is Nothing
```

Remarks

See Also

Shell, ShellExec, Exec, WinExec

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

Exec Function

Purpose

Loads and runs a program.

Syntax

% = Exec(file$, CmdLine)(function)

Exec file$, CmdLine(command)

file$, CmdLine:sexp

Description

The **Exec**(file$, CmdLine) function invokes program *file$* and gives it the command line *CmdLine*.

The *file$* expression contains the name of the called program. The program name includes the full pathname.

The *CmdLine* expression contains the command line which is inserted in the program segment prefix of the called program.

Example

```
Global a%
If Exist(WinDir + "\notepad.exe")
  a% = Exec(WinDir + "\notepad.exe", "")
  Message "return value: " & Format(a%)
Else
  Message "Program not found"
EndIf
```

```
Do
  Sleep
Until Me Is Nothing
```

Remarks

Exec internally uses the function **WinExec**(). If you want to determine, if the called program is still active or not, you must use **System** instead, which returns the handle of the process and allows controlling it. Also, **System** allows, by using the parameter **Wait**, to wait until the program has finished.

See Also

Shell, ShellExec, System, WinExec

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

WinExec Function

Purpose

Loads and runs a program.

Syntax

% = WinExec(file, CmdShow)(function)

WinExec file, CmdShow(command)

file:sexp
CmdShow:iexp

Description

The **WinExec**(*file, CmdShow*) function invokes program *file.*

The *file* expression contains the name of the called program. The program name includes the full pathname and the command line which is inserted in the program segment prefix of the called program.

CmdShow specifies the visual aspect of the window and is one of the SW_ constants SW_NORMAL, SW_HIDE, SW_SHOW, ... See **ShowW**.

Example

```
Global a%
If Exist(WinDir + "\notepad.exe")
  a% = WinExec(WinDir + "\notepad.exe", SW_NORMAL)
  Message "return value: " & Format(a%)
```

```
Else
  Message "Program not found"
EndIf
Do
  Sleep
Until Me Is Nothing
```

Remarks

If you want to determine, if the called program is still active or not, you must use **System** instead, which returns the handle of the process and allows controlling it. Also, **System** allows, by using the parameter **Wait**, to wait until the program has finished.

See Also

Shell, ShellExec, System, WinExec

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

ShellExec Function

Purpose

Opens, explores, or prints a specified file or folder.

Syntax

x% = **ShellExec**(file$ [, parameters$][, directory$])

x% = **ShellExec([**operation$][, file$][, parameters$][, directory$] [,show%])

ShellExec file$ [, parameters$][, directory$]

ShellExec [operation$][, file$][, parameters$][, directory$] [,show%]

Description

Returns a value greater than 32 if successful, or an error value that is less than or equal to 32 otherwise.

ShellExec uses the Window handle of the current active Form. This window receives any message boxes that an application produces. For example, an application may report an error by producing a message box. Null is passed when IsNothing(Me) is true.

All arguments are optional, but to differentiate between the three parameter and the five parameter version the five parameters version must be made explicit, by including enough (3) comma's. To use the five parameter version, you could use:

ShellExec ,file$, , [,]

operation$ specifies the operation to perform. The following operation strings are valid:

> "open" - The function opens the file specified by the *File$* parameter. The file can be an executable file or a document file. It can also be a folder.
>
> "print" - The function prints the file specified by *File$*. The file should be a document file. If the file is an executable file, the function opens the file, as if "open" had been specified.
>
> "explore" - The function explores the folder specified by *File$*.

When this parameter is omitted, NULL is passed. In that case, the function opens the file specified by *File$*. To open the Window Explorer use the five parameter version: **ShellExec** "explore", ".", ,.

file$ specifies the file to open or print or the folder to open or explore. The function can open an executable file or a document file. The function can print a document file. If *file$* specifies a document file, *show* should be zero. Use the three parameter version.

parameters$ specifies the parameters to be passed to the application, when the *File$* parameter specifies an executable file. If *File$* specifies a document file, *Parameters$ should omitted.*

directory$ specifies the default directory.

show specifies how the application is to be shown when it is opened. This is one of the SW_ constants, see **ShowW**.

Example

```
ShellExec "", "notepad", , , SW_MAXIMIZE
~ShellExec("explore", "d:", , , )
```

Remarks

See Also

Exec, Shell, System, WinExec

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Mci$ Function

Purpose

Executes a Mci (Multimedia Control Interface) command.

Syntax

**err = Mci[$]**(cmd$ [, formvar])

cmd$sexp
wininteger expression

Description

Mci$**(*cmd$*) executes a Mci command (as **Mci$("status id mode")). An error (in the command string, or any other error) is not reported with a message box, but returned as result *err*. (-1 if the mmsystem could not be found).

With **mciErr$**(*err*) you get the descriptive error text which would have been displayed for the **Mci** command.

Mci$**(*cmd*$, *formvar*) Does the same as **Mci$(sexp). The window (form object) given in integer expression (Win_1) gets a MM_MCINOTIFY message ($3b9) when the mci command finished execution (dummy$=**Mci$**("play id notify",Win_1)).

The MM_MCINOTIFY message can be handled in Win_1_**MciNotify**(devID%, Code%) event sub.

Remarks

The MM_MCINOTIFY message ($3b9). The Code% is returned in wParam.

wParam=1 - Mci command aborted

wParam=2 - Mci command successful

wParam=4 - Mci superseded by a new notify command

wParam=8 - Mci error, not reported when using **Mci$**()

LoWord(lParam) = Device ID (devID%) sending the message.

(the notify message is not sent, if the Mci returned an error in _EAX.)

See Also

Mci, mciErr$, mciID

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Mci Command

Purpose

Executes a Mci (Multimedia Control Interface) command.

Syntax

Mci cmd$

cmd$sexp

Description

Executes a **Mci** command (as **Mci** "close all"). An error (in the command string, or any other error) is reported with a message box. Error free execution set _**EAX** to 1. GFA-BASIC sets _**EAX** to -1 if the mmsystem could not be found.

Remarks

Multimedia is supported in Windows 3.x as a MMSYSTEM.DLL. It allows the handling of sound cards, audio CD-ROMs, videodisks, overlay video and animation etc. The lowest system of multimedia programming is by directly calling the device drivers for each device. Even the device independent programming interface of the multimedia system is quite complicated. There are different layers of multimedia support routines inside the mmsystem. The mmio system is a low level system for accessing multimedia files, it can only be used if the file structure is very well known, and is intended to be used in low level recording and manipulation systems and, as such, provides little help for standard presentation programs. Above this is the Mci system. This Multimedia Control Interface provides all the routines to access the devices in an orderly way. It's possible to use a message based system, with *mciSendMessage*, but this leads to hard to read code. The Mci provides a string based system, with *mciExecute* and *mciSendString*, which allows readable strings to be used for communication with the devices (as in "play tune", "seek to start"). This is the system chosen for GFA-BASIC. Multimedia is supported in GFA-BASIC for Windows version 4.22 upwards.

The command **Mci** and the function **Mci$** handle all the Mci operations, supported by **mciErr**$ and **mciID** for error text and a device id used for the notification message.

The Mci command strings are all used as "cmd id param", optional followed by "notify" or "wait".

cmd is one of the command words as open, play, record ...

id is the ID of a device. That can be one of the following:

Sequencer	MIDI or AdLib-Sound (build into virtually all sound cards)
WaveAudio	The usual sampled sound (voice, digital sound effects ...)
CDAudio	A Sound playing CD-ROM (attached to a sound card)
Videodisc	A Video CD
Overlay	An overlay of video images onto the computer screen
Animation	The Movie, similar to overlay, but the "video" is computer generated as well, from a (compressed) file.
	There is plenty of expansion

possible, like mmmovie for microsoft's animation. Most of the time, the device name is only used with the "capability" and "info" verbs to get information on the device without opening it. The "open" does support an "alias name". This let's you define a name to reference the device (and files) in a more abstract way. It allows to reference different files with the same, short identification.

param is an, often optional, parameter, or list of parameters. For open this is usually at least the "alias id".

notify optional following all commands is notify. If used, usually in a play, record or seek, a message (_Mess = $3b9) is send to a window The Handle has to be given as the optional second parameter to Mci$()

wait optional following all commands is wait. If used, the Mci function waits for completion.

The functions returning some value, as "capability", "info", "status", "sysinfo" or "where" are always used in Mci$(). There is exactly one parameter (as "status id length"), it returns a string (as "12340", "12:59:30:72", "true"). Commands, as "set", usually accept several parameters in one call (as "set id samplespersec 11025 bitspersample 8 time format ms channels 1"). The commands may be used with the GFA-BASIC command **Mci** or the function **Mci$().**

MCI Commands

In the list the character "**{**", "**}**", "[", "]" and "|" have a special meaning.

A string in [] is optional (without the []).

A | marks alternatives (one of a group of strings).

A string group in **{}**, separated by | means one of the strings in the **{}** is required, but only one..

Examples:

[insert | overwrite]:-> "insert" or "overwrite", or "".

{ to end | to start **}**:-> "to start" or "to end"

[a] [b] [c]:-> "", "a", "b", "a b", "c b a" or "c" or ...

A % is a place holder for a number (123) or a time (depending on time format). A group of four % % % % is a rectangle (example: "100 80 400 120" := left 100, top=80, width=400, height=120). A $ is a string, a series of characters (TestTitel). Optionally it can be enclosed in quotation marks ("Test Title") to allow spaces in the string.

Time formats are used in position, to % or from%. There are several time formats defined, to be selected with "set id time format $".

	time	**format**	**Position is**
millisecond	2000	2 seconds	
ms	2000	2 seconds	
msf	23:40:23	minute : second : frame	0-99:0-59:0-74

tmsf	3:23:40:23	track : minute : second : frame	0-99:0-99:0-59:0-74)
hms	23:59:59	hour : minute : second	
frames	2728	frame	2728
bytes	2700	byte	no 2700
samples	2700	sample	no 2700
track	3	track	3
song pointer	32	sixteenth notes	note 2
SMPTE x	02:12:0:08	hour : minute : second : frame (MIDI specific)	

System Commands

break id **{** on % | off **}**

sysinfo id **{** installname | quantity | quantity open | name % | name % open **}**

Required Commands

capability id **{** can eject | can play | can record | can save | uses files **}**

capability id **{** compound device | device type | has audio | has video **}** close **{** id | all **}**

info id product

open device[!file] [alias $id] [shareable] [type
$device_type]

status id mode

Basic Commands

load dev [filename]

pause id

play id [from %] [to %]

record id [insert | overwrite] [from %] [to %]

resume id

save id [filename]

seek id **{** to % | to start | to end **}**

set id **{** audio all off | audio all on | audio left off | audio left
on **}**

set id **{** audio right off | audio right on | door closed | door
open **}**

set id **{** video off | video on | time format millisecond | time
format ms **}**

status id **{** current track | length | length track % | ready |
start position**}**

status id **{** number of tracks | position | position track % | time format **}**

stop id

Animation Commands

capability id **{** can reverse | can save | can stretch | fast play rate **}**

capability id **{** normal play rate | slow play rate | uses palette | windows **}**

info id **{** file | window text **}**

open id [nostatic] [parent %] [style **{** % | child | overlapped | popup **}**]

play id [fast] [reverse] [scan] [slow] [speed %]

put id **{** destination | source **}** [at % % % %]

realize id **{** background | normal **}**

set id time format frames

status id **{** forward | media present | palette handle | speed | stretch **}**

status id **{** time format | window handle **}**

step id [by %] [reverse]

update id hdc % [at % % % %]

where **{** destination | source **}**

window id [fixed] [handle %] [handle default] [state hide] [state iconic] [state maximized]

window id [state minimize] [state minimized] [state no Purpose] [state no activate]

window id [state normal] [state show] [stretch] [text $]

Cdaudio Commands

set id time format **{** msf | tmsf **}**

Sequencer Commands (midi)

info id file

save id [filename]

set id [master MIDI] [master none] [master SMPTE] [offset %] [port %] [port mapper]

set id [port none] [slave file] [slave MIDI] [slave none] [slave SMPTE] [tempo %]

set id [time format song pointer] [time format SMPTE 24] [time format SMPTE 25]

set if [time format SMPTE 30] [time format SMPTE 30 drop]

status id **{** division type | master | offset | port | slave | tempo **}**

Videodisc Commands

capability id **{** CAV | CLV **}**

escape id $

seek id reverse

set id [time format hms] [time format track]

spin id **{** up | down **}**

status id **{** disc size | forward | media type | side **}**

set id [by % | by % reverse | | reverse | by -%]

Overlay Commands

capability id windows

freeze id [at % % % %]

info id window text

load id [filename] [at % % % %]

put id [video [at % % % %]] [frame [at % % % %]]

put id [source [at % % % %]] [destination [at % % % %]]

save id filename [at % % % %]

unfreeze id [at % % % %]

where id **{** video | frame **}**

Waveaudio Commands

capability id **{** inputs | outputs **}**

cue id **{** input | output **}**

delete id [from %] [to %]

info id **{** input | output **}**

open ... [buffer %]

open new type waveaudio ...

set id [alignment %] [any input] [any output] [bitspersample %]

set id [bytespersec %] [channels %] [format tag $] [format tag pcm]

set id [input %] [output %] [time format bytes] [time format samples]

status id **{** alignment | bitspersample | bytespersec | channels | format tag **}**

status id **{** evel | input | output | samplespersec **}**

Important: The Mci does not work for a synchronous wave device. That is the PC speaker driver from Microsoft. The speaker driver does only work with PlaySound.

Example

```
// play alarm01.wav three times
// first version checks for end of sound playing
  with the "status id mode" function.
// second version checks using the notify flag,
  and is about 30 times faster.
// If MCI can not find the above files, change the
  addresses to files on your local machine.
Auto i%, q%
OpenW # 1
```

```
Mci "open c:\windows\media\alarm01.wav alias bong"
For i% = 1 To 3
  Mci "play bong from 1"
  q% = 0
  Do
    PeekEvent
    q%++
  Loop Until Mci$("status bong mode") != "playing"
  Print q%
Next i%
Mci "close bong"
Mci "open c:\windows\media\alarm01.wav alias bong"
For i% = 1 To 3
  ~Len(Mci$("play bong from 1 notify"))
  If _EAX = 0       //simple error check
    q% = 0
    Do
      PeekEvent
      q%++
    Loop Until _Mess = $3b9
    Print q%
  EndIf
Next i%
Mci "close bong"
```

See Also

Mci$, mciErr$, mciID

Microsoft Developer Network

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

mciErr$ Function

Purpose

Gets the descriptive text for a Mci error.

Syntax

mciErr$(errno)

errnointeger expression

Description

Gets a Description of an Mci error as text. This is the text which is displayed when using the Mci command in a message box. The error code (integer expression) is returned from Mci$().

Example

```
// prints the Mci error message
// are in the range of 0 till 32767
Debug.Show
Local a$, a%, i&
// only a part are filled with usable error
 messages
// please test it by yurself, if necessary
For i& = 0 To 32767
  a$ = mciErr$(i&)
  If Len(a$) <> 0 // if error i& exist
    Debug "Error:";i&, a$ // print Mci error
  EndIf
Next i&
```

See Also

Mci, Mci$, mciErr$, mciID

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

mciID Function

Purpose

Returns the ID for an opened Mci device

Syntax

mciID(name$)

name$sexp

Description

This function returns the ID for an opened device. Usually used with an alias name. Used to get the device id for the notify message.

Example

```
Debug.Show
Mci "open c:\windows\media\alarm01.wav alias bong"
Trace mciID("bong")
Mci "close bong"
Trace mciID("bong")
```

See Also

Mci, Mci$, mciID

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

SelPrint, SelPrintRect Methods

Purpose

Sends formatted text in a **RichEdit** control to a device for printing.

Syntax

RichEdit.**SelPrint(***hDc***)**

RichEdit.**SelPrintRect(***hDc, l, t, w, h***)**

hDc:Handle
l, t, w, h:Single exp

Description

If text is selected in the **RichEdit** control, the **SelPrint** method sends only the selected text to the target device. If no text is selected, the entire contents of the **RichEdit** are sent to the target device.

The **SelPrint** method does not print text from the **RichEdit** control. Rather, it sends a copy of formatted text to a device which can print the text.

SelPrintRect(*hDc, l, t, w, h)* prints a portion of a rich edit control's contents, as previously formatted for a device *hDc*, to a rectangle area of that device. The rectangle is specified in twips with l (left), t (top), w (width), and h (height) parameters. The returns value of **SelPrintRect** is the index of the first character that doesn't fit the rectangle.

Example

```
Lprint "";
rtf1.SelPrint(Printer.hDC)
```

Example 2

```
StartDoc "Test"
StartPage
rtf1.SelPrintRect(Printer.hDC, 0, 0, 2000, 2000)
EndPage
EndDoc
```

Remarks

If you use the **Printer** object as the destination of the text from the **RichEdit** control, you must first initialize the device context of the **Printer** object by printing something like a zero-length string.

Known Issues

Problems have been reported with both **SelPrint** and **SelPrintRect** either just not printing or, more seriously, causing the program to freeze. There are currently no workarounds to these problems.

See Also

RichEdit, FormatDC, Printer

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Beep Command

Purpose

Sounds a warning.

Syntax

Beep

Description

Sounds a short beep on the system speaker

Example

```
Beep
```

Remarks

This command corresponds to the Windows function *MessageBeep*().

See Also

PlaySound

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

PlaySound Command

Purpose

Play a WAV-file.

Syntax

PlaySound wav$ [, flag = 0]

wav$sexp
flagiexp

Description

The **PlaySound** command plays the WAV (-file) requested by the user. *wav$* may specify a filename or a string containing WAV data. If wav$ = "", any currently playing waveform sound is stopped.

flag	**Meaning**
SND_SYNC (0)	the sound plays synchronously and waits untill the playing event ends.
SND_ASYNC (1)	The sound starts asynchronously and immediately returns to the program (doesn't wait).
SND_NODEFAULT (2)	when the sound file cannot be found, the function returns to the program without playing a predefined default sound (usually a warning)
SND_MEMORY (4)	A sound is started whose file is loaded in string memory.
SND_LOOP (8)	The sound plays repeatedly until

	PlaySound "" is called You must also specify the SND_ASYNC flag to indicate an asynchronous sound event.
SND_NOSTOP (16)	if another song is just being played; the new sound is put in a queue and will be played after completion of the current sound.

If it cannot find the specified sound, **PlaySound** uses the default system event sound entry instead.

Example

```
OpenW 1
Print "Playing Tada.wav"
PlaySound WinDir + "\media\tada.wav", SND_SYNC
Print "Playing Notify.wav"
PlaySound WinDir + "\media\notify.wav", SND_NOSTOP
```

Remarks

The **PlaySound** command uses the installed sound-driver.

See Also

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Stick Function

Purpose

Reads joystick or touch screen position.

Syntax

% = **Stick**(n)

Description

Stick returns the position of the multi-media input device in pixels. The range is from 0 to 65535, from the left-top to the right-bottom.

Stick(0) - Reads the horizontal position (x-coordinate) of the joystick #1

Stick(1) - Reads the vertical position (y-coordinate) of the joystick #1

Stick(2) - Reads the horizontal position (x-coordinate) of the joystick #2

Stick(3) - Reads the vertical position (y-coordinate) of the joystick #2

Stick(1) .. **Stick**(3) are the positions stored at the time of the last **Stick**(0). That means that a **Stick**(0) is needed to really read both sticks positions.

Example

```
~Stick(0)
Print Stick(2), Stick(3)
```

See Also

Strig

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Strig Function

Purpose

Reads joystick buttons or other multi-media input devices.

Syntax

Bool = **Strig**(n)

Description

Strig(0) - Checks if the first button of the first joystick has been pressed.

Strig(1) - Checks if the first button of the first joystick is currently being pressed

Strig(2) - Checks if the first button of second joystick has been pressed.

Strig(3) - Checks if the first button of the second joystick is currently being pressed.

Strig(4) - Checks if the second button of the first joystick has been pressed.

Strig(5) - Checks if the second button of the first joystick is currently being pressed

Strig(6) - Checks if the second button of second joystick has been pressed.

Strig(7) - Checks if the second button of the second joystick is currently being pressed.

The odd numbers return -1 if the corresponding buttons are held down. The even numbers return -1 only once if the buttons are just pressed down, then they return 0.

Example

```
// Mousek added to prevent infintie loops...
// ...if no joystick plugged in or...
// ...joystick not working correctly.
Auto x_bot%, x_mid%, x_top%, y_bot%, y_mid%,
  y_top%
Print "Centre and Click:"
Repeat
  x_mid% = Stick(0), y_mid% = Stick(1)
Until Strig(0) Or MouseK = 1
Print "Top/Left And Click:"
Repeat
  x_top% = Stick(0), y_top% = Stick(1)
Until Strig(0) Or MouseK = 2
Print "Bottom/Right And Click:"
Repeat
  x_bot% = Stick(0), y_bot% = Stick(1)
Until Strig(0) Or MouseK = 1
```

Remarks

The Joystick functions use the Windows multi media functions (joyGetPos) to read the joystick, not the interrupts as in GFA-BASIC 16.

See Also

Stick

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Environ Function

Purpose

Returns and sets the value of an operating system environment variable.

Syntax

Environ[$]("name" | number) [= value$]

Description

Environment variables define the environment in which a process executes (for example, the default search path for libraries to be linked with a program).

If "name" can't be found in the environment-string table, a zero-length string ("") is returned. Otherwise, **Environ** returns the text assigned to the specified "name"; that is, the text following the equal sign (=) in the environment-string table for that environment variable.

If you specify *number*, the string occupying that numeric position in the environment-string table is returned. In this case, **Environ** returns all of the text, including the name. If there is no environment string in the specified position, **Environ** returns a zero-length string.

Environ("name" | number) = creates new environment variables; modifies or removes existing ones.

Example

```
Environ("CopyOfPath") = Environ("Path")
Environ("Dircmd") = "/4"
Debug.Show
// path out of the Autoexec.dos
Trace Environ("path")
Trace Environ("comspec")
// more
Trace Environ("TEMP")
Trace Environ("TMP")
Trace Environ(1)
Trace Environ(2)
Trace Environ(14)
Trace Environ(15)
```

Remarks

See Also

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

IsWinNT Function

Purpose

Helps in differentiating between Windows 95, 98, and Me versus Windows NT, 2000, and XP.

Syntax

Bool = **IsWinNT**

Description

The main difference between Windows 95, 98, and Me and the real 32 bit versions NT, 2000, and XP is the Win API version. The 32-bits version support the Win API 4.0.

Example

```
Print IsWinNT
```

Remarks

IsWinNT is the same as *GetVersion*() >= 0.

The Windows API function *GetVersion*() returns a positive number for Windows NT, and a negative for Windows 95/98.

See Also

WinVersion

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

_CmdLine Function

Purpose

Returns the command line.

Syntax

$ =**_CmdLine**[$]

Description

_CmdLine returns the MS-DOS or Windows command line; the filename and the command line parameters passed when the program is started. Used in the IDE the **_CmdLine** returns the IDE name including its full path.

Example

```
Print _CmdLine
```

Remarks

_CmdLine is equivalent with Print Char{GetCommandLine()}

_CmdLine is the only function that includes the program's filename (full path).

See Also

_DosCmd$, Arguments

{Created by Sjouke Hamstra; Last updated: 18/09/2014 by James Gaite}

_DosCmd$ Function, Arguments Property (App)

Purpose

Returns the MS-DOS or MS-Windows command string (from the command line).

Syntax

$ = **_DosCmd**$

$ = App.**Arguments**

Description

These functions return the arguments of the program without the name of the program.

Example

```
Global Const __argmax = 50
Global __argc As Int
Global __argv() As String
Local Int32 n
ConvertCMDLine()
Print __argc
Print __argv(0)
For n = 1 To __argc : Print __argv(n) : Next n
Do : Sleep : Until Me Is Nothing

Procedure ConvertCMDLine()
  // The global variable __argc holds the
// number of commandline arguments after
  executing ConvertCMDLine().
// Arguments are separated by space(s)
// __argv() is an Array with the split arguments.
// Only __argmax arguments are returned.
// __argv(0) holds the complete path, filename
  included.
// Note: This routine can not differentiate
  between spaces in filenames
//       and spaces separating arguments.
Local i As Int = 0, j As Int = 0
Local cmd$
Local LargeArg As Boolean = False
Local a$
ReDim __argv(__argmax)
__argv(0) = App.FileName
// Remove quotes
If Left$(__argv(0), 1) = #34
  __argv(0) = Mid(__argv(0), 2)
EndIf
If Right$(__argv(0), 1) = #34
  __argv(0) = Left(__argv(0), Len(__argv(0)) - 1)
EndIf
cmd$ = Trim(_DosCmd$) + #32
If Left$(cmd$, 1) <> """"
  i = InStr(cmd$, #32)
Else
  Debug.Print InStr(cmd$, """", 2)
  i = InStr(cmd$, #34, 2) : LargeArg = True
  cmd$ = Mid$(cmd$, 2) // remove space at start
EndIf
While i > 0
  j++
  If LargeArg
    // remove space at end
    a$ = Left$(cmd$, i - 2)
    If Len(a)
```

```
        __argv(j) = Left$(cmd$, i - 2)
      Else
        j--
      EndIf
      LargeArg = False
    Else
      // only remove space at end
      a$ = Left$(cmd$, i - 1)
      If Len(a)
        __argv(j) = Left$(cmd$, i - 1)
      Else
        j--
      EndIf
    EndIf
    Exit If (i + 1) > Len(cmd$)
    cmd$ = Mid$(cmd$, i + 1)
    If Left$(cmd$, 1) <> """"
      i = InStr(cmd$, #32)
    Else
      i = InStr(cmd$, #34, 2) : LargeArg = True
      cmd$ = Mid$(cmd$, 2) // remove space at
        beginning
    EndIf
  Wend
  // Return number of arguments
  __argc = j
EndProcedure
```

The above routine only works if the path does not contain spaces.

Remarks

_DosCmd and **App.Arguments** only provide the command line parameters. In contrast, **_CmdLine** also includes the program's full path name.

See Also

_CmdLine

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

Asm Command

Purpose

Invokes the inline assembler.

Syntax

. | Asm *mnemonic destination, source*

Description

The inline assembler lets you embed assembly-language instructions directly in your GFA programs without extra assembly and link steps. The inline assembler is built into the compiler - you don't need a separate assembler such as the Microsoft Macro Assembler (MASM).

Because the inline assembler doesn't require separate assembly and link steps, it is more convenient than a separate assembler. Inline assembly code can use any GFABASIC32 variable or functionname that is in scope, so it is easy to integrate it with your program's code. And because the assembly code can be mixed with other statements, it can do tasks that are cumbersome or impossible in GFABASIC alone.

The dot is a shortcut for the **Asm** keyword and invokes the inline assembler and can appear wherever a GFA-BASIC 32 statement is legal. It cannot appear by itself. It must be followed by an assembly instruction.

The assembler commands use the INTEL parameter sequence, for example:

. mov dest, source

The following code consists of simple Asm block. (The code is a custom function prolog sequence.)

```
Asm push ebp
Asm mov  ebp, esp
Asm sub  esp, __LOCAL_SIZE
```

Alternatively, you can use a 'dot - space' in front of each assembly instruction:

```
. push ebp
. mov  ebp, esp
. sub  esp, __LOCAL_SIZE
```

You can also put assembly instructions on the same line using the statement separator:

```
. nop : . inc eax
Asm nop : Asm inc eax
```

Assembler labels

A label inside an assembler block differs from the rules above. A label always starts with dot directly followed by its name, and directly followed by a semicolon (:). However the semicolon is not used in the jump or call instruction.

```
. cmp eax, 0
. je .next
. jmp .exit
.next:
. cmp eax, 65
```

Like an ordinary GFA-BASIC 32 label, an assembler label has scope throughout the function in which it is defined. Both assembly instructions and GoTo/GoSub statements can jump to labels inside or outside the assembler instructions. GoTo and GoSub refer to the assembler labels without the preceding dot.

To jump to an ordinary label using an assembler instruction, the label is preceded with a dot when used as an argument in the instruction.

```
Dim i As Int
test:' GFA-BASIC 32 label
Print "Hallo"  : i --
. mov eax, [i]
. test eax, eax
. je .test; note the dot in front of the label
```

To jump to an assembler label using GoTo or GoSub leave out the starting dot.

GoTo 00Assem' note the missing dot

```
// assembler code
.00Assem: Print "Hallo"
```

Because assembler label names start with a dot, GFA-BASIC 32 allows the use keywords for label names, this in contrast to C/C++ inline assembler. This feature allow you to choose meaningful label names like .next, .try, .exit, .end, that would otherwise impossible.

Using variables

To move the contents of a variable to a register use mov reg, [varname]. For instance

```
Dim i As Int, j%
. mov eax, [i]
. mov [j%], eax      ; $AutoPost has no meaning
```

$AutoPost settings are not obliged in assembler instructions. This can be a point for confusion; in assembler i As Int is different from i%.

The variable name is a place holder for the address of the variable; the instruction mov eax, [i] is the same operation as **DPeek**(*i).

The compiler directly inserts the address of the variable for the second argument [i]

```
. mov eax, [$00EEDD11]
```

For local variables, the address of the variable is compiled as an offset from ebx.

```
. movzx eax, [localvar]
. movzx eax, wpt [ebx + 124] ; if it is the first local variable
```

Using the form mov reg,[var] to get access to the value of the variable is only true for simple types like integers and floating-point data types. The address of the variable is the place where the data is kept. To access variables this way the following must be valid: ***var** = **ArrPtr**(var) = **V:** var

The fixed string and UDT data types are accessible through *, as well. For example a fixed string can be indexed as follows

```
Debug.Show
f()

Sub f()
  Dim sFixed As String * 26
  .   xor ecx, ecx
  .   mov al, 65
  .l:
  .    mov sFixed[ ecx], al
  .    inc ecx
  .    inc eax
  .    cmp ecx, 25
  .    jle .l
  Trace sFixed
```

More complex type variables like strings and arrays are managed through their descriptor (*str <> V: str). The variable name is placeholder for a reference to their descriptor and not to the bytes where the actual data reside. For these types the starting address of the data bytes must be stored in a temporarily long integer (variable or register) and accordingly used.

The following example illustrates how to use variables by calling API functions with assembler.

```
' By John Findlay
Type RECT
```

```
  Left   As Long
  Top    As Long
  Right  As Long
  Bottom As Long
End Type
Global rc As RECT, hUser32 As Handle, i As Int
Global lpGetClientRect  As Int, lpGetWindowRect As Int
' Find the addresses of the two functions
hUser32 = LoadLibrary("user32.dll")
lpGetClientRect = GetProcAddress(hUser32, "GetClientRect")
lpGetWindowRect = GetProcAddress(hUser32, "GetWindowRect")
OpenW 1
Print "Example of calling the GetClientRect() and GetWindowRect()
  API's with assembler."
Print
Print "Client Coords"
Print MyGetClientRect(Win_1.hWnd, *rc), "Return value from asm call"
Print rc.Left,   "Left"
Print rc.Top,    "Top"
Print rc.Right,  "Right"
Print rc.Bottom, "Bottom"
Print
Print "Window Coords"
Print MyGetWindowRect(Win_1.hWnd, rc), "Return value from asm call"
Print rc.Left,   "Left"
Print rc.Top,    "Top"
Print rc.Right,  "Right"
Print rc.Bottom, "Bottom"
Print
Print "Press a key to exit."
~FreeLibrary(hUser32)
KeyGet i
CloseW 1

Function  MyGetClientRect(hWnd As Int, lpRect As Int) As Int Naked
  . push [lpRect] : . push [hWnd]
  . call [lpGetClientRect]
  . mov [MyGetClientRect], eax  ' Return
EndFunc

Function  MyGetWindowRect(hWnd As Int, ByRef lpRect As RECT) As Int
  . push [lpRect] : . push [hWnd]
  . call [lpGetWindowRect]
  . mov [MyGetWindowRect], eax  ' Return
EndFunc
```

Note The mov instructions in both functions are redundant. Return values from functions (API or GFA-BASIC 32) are always placed in eax. Returning a value through a temporary variable with the same name as the function name is a VB quirk, which simply results in a move back to eax, which then holds the return value of the function.

Assembler data

To define constant values the following assembler statements are available

. *db* const - byte constants and Strings (values -128 to +255)

. *dw* const - 2 byte integer (-32768 to + 65535)

. *dd* const - 4 byte integer (possible to store label address)

. *dl* const - 8 byte (large) integer

. *dq* const - 8 byte (double) floating-point (. dq 12.34 or . dq PI/180*23.5)

. *ds* const - 4 byte (single) floating point (. ds 12.34 or . ds 12.34!)

In contrast with other assemblers . dd 1.0 is not the same as . ds 1.0!

Examples:

```
.text:
. db "This is a Text", 0
. dd "This is a Text", 0
```

Shortcuts

Out of efficiency reasons, there are shortcuts for *byte ptr, word ptr, dword ptr, qword ptr, tbyte ptr,* and *fword ptr.* The shortcuts are respectively, *bpt, wpt, dpt, qpt, tpt, and fpt*. The following instructions are equivalent.

```
. mov bpt [i], 1
. mov byte ptr [i], 1
```

The disassembler uses the shortcuts by default (cannot be changed).

Jumping and calling

The jump statements (jcc, jmp, loopx, jcxz, etc) only accept a relative offset or a label:

```
. jc $+nn; addresss relative to $ ( = eip )
. jmp $+ 2
. jc .label; a label (use .)
```

As in MASM programs, the dollar symbol (**$**) serves as the current location counter. It is a label for the instruction currently being assembled.

With call and jmp other addressing modes are possible as well:

```
. call ecx
. jmp .tab[eax*4]
```

Calling GFA-BASIC 32 functions

A special assembler command - **scall** - is required to call a GFA-BASIC 32 function by its name. For instance, to call the GFA-BASIC 32-internal function MessageBeep(0) the following is used:

```
. push 0
. scall MessageBeep
```

Internally, scall is implemented as *call dword ptr [ ]* , where the name is known to the compiler only.

Floating-point extensions

GFA-BASIC 32 extents the normal INTEL x86 floating-point assembler instructions that works with a constant. For instance, there is no command like fadd 0.125, instead (external) assembler requires the following construction:

```
'data
Kon0_125: . dd 0.125
'text
. fadd [Kon0_125]
```

The GFA-BASIC 32 inline assembler allows simple additions like this:

```
. fadd 0.125
```

The management of the memory for the constants is done by the assembler.

The floating-point instructions fld, fadd, fsub, etc. support both Double (default) and Single arguments. To force single floating-point operations the argument must be converted to a single value explicitly like

```
. fadd 12.4!
. fadd CSng(PI)
```

The floating-point extensions apply to *fadd, fsub, fmul, fdiv, fsubr, fdivr, fcom, fcomp, fldcw, fld, fild, fbld, fiadd, fisub, fisubr, fimul, fidiv, fidivr, ficom, ficomp*.

(The integer statement bound (bound eax, [addr] or bound eax, lo, hi) also accepts a constant.)

Math with labels

The difference between two labels can be obtained indirectly only:

```
. mov ecx, .label2
. sub ecx, .label1
```

Since math with label addresses is forbidden, the following is not allowed:

```
. mov ecx, .label2 - .label1; not possible
. mov al,[.label][3]
```

Rather than:

```
.tmp: . dd 1234
. mov al,[.tmp][3]; not allowed
```

you should use:

```
.tmp:  . db GetByte0(1234)
.1tmp: . db GetByte1(1234)
.2tmp: . db GetByte2(1234)
.3tmp: . db GetByte3(1234)
. mov al,[.3tmp]
```

or better:

```
.tmp: . dd 1234
. lea eax,[.tmp]
. mov al, 3[eax]
```

This restriction applies to labels only and not to variables:

```
Dim iTmp As Int
. mov al, 3[iTmp]
```

Assembler Opcodes

To identify commands that require a 80486-processor the first character is uppercase (this is automatically set by the editor). For instance .Xadd and .Cmpxchg require processors with at least a 80486 processor and are visually identified by their uppercase.

```
. Xadd [i], eax
. Cmpxchg [eax], ecx
```

To identify Pentium statements the first two characters are converted to uppercase. For instance, the GFA-BASIC 32 function _Rdtsc requires a Pentium and should it used in assembler it is visually differentiated.

```
. RDtsc
. Cmpxchg qpt [i]
```

Finally, MMX commands like PADD are entirely uppercase.

Assembler statements

aaa	**aad *10**	**aam *10**	**aas**	**Adc**	**add**
align † N	and	arpl	bound	Bsf	bsr
Bswap	bt	btc	btr	Bts	call
cbw	cdq	clc	cld	Cli	clts
cmc	cmp	cmps	cmpsb$^{\circ\circ}$	Cmpsd$^{\circ\circ}$	cmpsw$^{\circ\circ}$
Cmpxchg	CMpxchg8b	CPuid	cwd	Cwde	daa
das	db † N	dd † N	dec	Div	dl † N
dq † N	ds † N	dw † N	enter	f2xm1	fabs
fadd	faddp	fbld	fbstp	Fchs	fclex
fcom	fcomp	fcompp	fcos	fdecstp	fdisi
fdiv	fdivp	fdivr	fdivrp	Feni	ffree
fiadd	ficom	ficomp	fidiv	Fidivr	fild
fimul	fincstp	finit	fist	Fistp	fisub
fisubr	fld	fld1	fldcw	Fldenv	fldl2e
fldl2t	fldlg2	fldln2	fldpi	Fldz	fmul
fmulp	fnclex	fninit	fnop	fnsave	fnstcw
fnstenv	fnstsw	fpatan	fprem	Fprem1	fptan
frndint	frstor	fsave	fscale	fsetpm	fsin
fsincos	fsqrt	fst	fstcw	Fstenv	fstp
fstsw	fsub	fsubp	fsubr	Fsubrp	ftst
fucom	fucomp	fucompp	fwait	Fxam	fxch
fxtract	fyl2x	fyl2xp1	hlt	Idiv	imul
in	inc	ins	insb	Insd	insw
int	into	Invd	Invlpg	Iret	ja
jae	jb	jbe	jc	jcxz^{1}	jcxzd^{32}
je	jecxz^{32}	jg	jge	Jl	jle
jmp	jna	jnae	jnb	Jnbe	jnc
jne	jng	jnge	jnl	Jnle	jno
jnp	jns	jnz	jo	Jp	jpe
jpo	js	jz	lahf	Lar	lds
lea	leave	les	lfs	Lgdt	lgs
lidt	lldt	lmsw	lock	Lods	lodsb$^{\circ\circ}$
lodsd$^{\circ\circ}$	lodsw$^{\circ\circ}$	loop^{32}	loopd^{32}	loopde^{32}	loopdne^{32}
loopdnz^{32}	loopdz^{32}	loope^{32}	looped^{32}	loopew^{1}	loopne^{32}
loopned^{32}	loopnew^{1}	loopnz^{32}	loopnzd^{32}	loopnzw^{1}	loopw^{1}
loopwe^{1}	loopwne^{1}	loopwnz^{1}	loopwz^{1}	loopz ^{32}	loopzd^{32}
loopzw^{1}	lsl	lss	ltr	Mov	movb$^{\circ\circ}$

movl°°	movs	movsb°°	movsd°°	movsw°°	movsx
movsxb°°	movsxw°°	movw°°	movzx	movzxb°°	movzxw°°
mul	neg	nop	not	Or	out
outs	outsb °°	outsd °°	outsw °°	Pop	popa^{1}
popad^{32}	popf^{1}	popfd^{32}	popw^{1}	Push	pusha^{1}
pushad^{32}	pushf^{1}	pushfd^{32}	pushw^{1}	Rcl	rcr
RDmsr	RDtsc	rep	repe	Repne	ret
retf	retn	rol	ror	RSm	sahf
sal	sar	sbb	scall	Scas	scasb °°
scasd°°	scasw°°	seta	setae	Setb	setbe
setc	sete	setg	setge	Setl	setle
setna	setnae	setnb	setnbe	Setnc	setne
setng	setnge	setnl	setnle	Steno	setnp
setns	setnz	seto	setp	Setpe	setpo
sets	setz	sgdt	shl	Shld	shr
shrd	sidt	sldt	smsw	Stc	std
sti	stos	stosb°°	stosd°°	Stosw°°	str
sub	test	verr	verw	Wait	Wb_invd
WRmsr	Xadd	xchg	xlat	Xlatb	xor

MMX statements

EMMS	**MOVD**	**MOVQ**	**PACKSSDW**	**PACKSSWB**	**PACKUSWB**
PADDB	PADDD	PADDSB	PADDSW	PADDUSB	PADDUSW
PADDW	PAND	PANDN	PCMPCGD	PCMPEQB	PCMPEQD
PCMPEQD	PCMPEQW	PCMPGTB	PCMPGTW	PMADDWD	PMULHW
PMULLW	POR	PSLLD	PSLLQ	PSLLW	PSRAD
PSRAW	PSRLD	PSRLQ	PSRLW	PSUBB	PSUBD
PSUBSB	PSUBSW	PSUBUSB	PSUBUSW	PSUBW	PUNPCKHBW
PUNPCKHDQ	PUNPCKHWD	PUNPCKLBW	PUNPCKLDQ	PUNPCKULWD	PXOR

Pentium specific assembler and disassembler statements

For Pentium Pro/II/III... an additional set of move statements is added. The presence of these statements is indicated by bit #15 in _CPUIDD.

cMOVo	**cMOVno**	**CMOVb**	**cMOVc**	**CMOVnae**	**cMOVnb**	**CMOVnc**	**cMOVdae**
cMOVz	cMOVe	CMOVnz	cMOVne	CMOVb	cMOVna	CMOVnb	cMOVa

				e		e	
cMOVs	cMOVns	CMOVp	cMOVpe	CMOVnp	cMOVpo	cMOVl	cMOVnge
cMOVnl	cMOVge	CMOVle	cMOVng	cMOVnle	cMOVg		

These move statements move bytes when a condition is met (o, no, b, etc.), like jcc or setcc. As destination only one of the eight possible general registers is allowed (esp included). Also allowed are the 16 bit registers (and addresses). The source operand cannot be a constant.

Explanation

[1] This is the 16 bit statement, loopw.

With pushw/popw: *pushw ds* is a 16 bit push of the ds register, *push ds* a 32 bit push. Instructions using segment registers, not allowed in flat mode, are handled as pseudo-32 bit registers by the processor. Therefore, a far call in 32 bit mode requires 8 bytes for a return address (4 bytes offset, 2 byte cs and 2 byte dummy to pad to 32 bit).

[32] This is 32 bit instruction: loop, loopd

°° With instructions taking multiple data types (like *movs*) the size of data type can be specified by using a postfix character (b, w, or d). Saves a bit of typing:

```
. movsd
. movs dword ptr es:[edi], dword ptr [esi]
```

Other shortcuts for *mov* and *movsx*/*movzx*:

```
. MOVD 8[ebp], 12
. mov dword ptr 8[ebp], 12
. movzxb eax,[eax]
. movzx eax, byte ptr [eax]
```

† Pseudo instructions using constants as parameter:

align 2 - Alignment at word border (or a *nop*)

align 4 - Alignment on DWORD border (or some *nop*s or other instructions that don't modify registers: *mov ecx, ecx* or *lea edx, 0[edx]*)

align 8 - Alignment on 8 byte border (useful together with dq)

align 16 - Alignment on 16 byte border

See Also

. Assembler Instruction

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

GetRegsCommand

Purpose

Copies the processor registers.

Syntax

GetRegs

Description

GetRegs copies the content of the processor registers to the pseudo register variables _**EAX**, _**EBX**, etc.

The pseudo register variables are used to inspect processor registers and to pass values to assembler routines.

_EAX, _EBX, _ECX, _EDX, _ESI, _EDI, _EBP, _ESP, _EFL, _EIP

_AX, _BX, _CX, _DX, _SI, _DI, _BP, _SP, _FL, _IP

_CS, _DS, _ES, _FS, _GS, _SS

_AH, _AL, _BH, _BL, _CH, _CL, _DH, _DL

The top eight are the pseudo register variables of GFA-BASIC 32. They can be filled using the **GetRegs** command. **GetRegs** copies the values and state of each register processor in its corresponding variable. The value of the eax register is copied to _EAX, the ecx register to _ECX, etc.

For four registers, eax, ebx, ecx, and edx, the LoWord, and the LoByte and HiByte of the LoWord can be read and set individually. The table below shows the meaning and the relationship of the pseudo register variables.

Register	**Lo Byte 8 bit**	**Hi Byte 8 bit**	**Lo Word 16 bit**	**Register 32 bit**
accumulations register	**_AL**	**_AH**	**_AX**	**_EAX**
base register	**_BL**	**_BH**	**_BX**	**_EBX**
count register	**_CL**	**_CH**	**_CX**	**_ECX**
data register	**_DL**	**_DH**	**_DX**	**_EDX**
source index register			**_SI**	**_ESI**
destination index register			**_DI**	**_EDI**
base pointer register			**_BP**	**_EBP**
stack register			**_SP**	**_ESP**
flag register			**_FL**	**_EFL**
extended instruction pointer				**_EIP**
only for the 16 bit operating			**_IP**	
code segment register			**_CS**	
daten segment register			**_DS**	
extra segment register			**_ES**	
extra segment register			**_FS**	

extra segment register	**_GS**
stack segment register	**_SS**
carry flag	**_CY**

When an assembler routine is invoked, the registers can be initialized at the calling. The **Call**(**X**) command allows passing values by pseudo register variable.

```
Call(addr) ( _EAX = 1, _ECX = 2)
```

Further more, some GFA-BASIC 32 commands return values in a pseudo variable. For instance **Dlg Font** returns the size in point in **_DX** and the font type in **_SI**. All Dlg-common dialog commands return an error condition in **_AX**.

Example

```
Dim cur As Currency
cur = 1000
GetRegs : Print _EAX, _EDX
```

output of two register after addition to one Currency variable

Remarks

You are free to use the pseudo variables as (global) variables to store temporarily information.

See Also

Tron, Call, CallX

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

_CPUID Function

Purpose

Returns processor information.

Syntax

a = **_CPUID**

a: large ivar

Description

Every processor has an internal register containing information about its type and manufacturer. The information block is 128 bits (16 bytes) in size.

Example

Print the **_CPUID**.

```
Message Hex$(_CPUID)
// prints a key for the processor type, for
  instance 52c
// in case of a normal Pentium
MsgBox (_CPUID And $f00) = $500 ? "Pentium" : "
  Other processor"
// prints Pentium in case of a Pentium, otherwise
  Other processor
Print Choose((_CPUID >> 8) And 15, "", "", "386",
  "486", _
  "Pentium", "Pentium2", "Pentium III")
// or
```

```
Print Btst(_CPUIDD, 15) // Pentium II or Pentium
  Pro by checking "fcmove..."
Do : Sleep : Until Me Is Nothing
```

Remarks

The 14 lowest bits of _**CPUID** return the CPU type. The following is true:

_**CPUID** %& 0x3000

0	Normal CPU
1	Overdrive CPU
2	Dual processor
3	Intel Reserved

_**CPUID** %& 0x0fff

300	
300	386 (no CPUID-assembler instruction)
4XX	
400	486 (no Cpuid-assembler instruction)
44x	486SL
47x	486DX2, WriteBack Enhanced
48x	486DX4 (or Overdrive)
5XX	
51x	Pentium 60 or 66 (or Overdrive)
52x	Pentium 75, 90, 100, 120, 133, 150, 166 or 200 (or Overdrive)
53x	Pentium Overdrive 486
54x	Pentium MMX 166/200
54x	Pentium MMX Overdrive 75/90/100/120/133
6XX	

61x	Pentium Pro
63x	Pentium II, Model 3
63x	Pentium II Overdrive
65x	Pentium II-5, Celeron-5, Pentium II-Xeon
66x	Celeron-6
67x	Pentium III, Pentium III-Xeon

The Pentium II, Model 5, and the Celerons, or Pentium II-Xeon can be separated by the 2nd Level Cache Information. The same is true for Pentium III and Pentium III-Xeon.

See Also

_CPUIDD, _CPUID$

{Created by Sjouke Hamstra; Last updated: 18/09/2014 by James Gaite}

_CPUID$ Function

Purpose

Returns the processor type as name.

Syntax

$ = **_CPUID$**

Description

The name of the CPU in plain text.

Example

```
MsgBox _CPUID$ // Returns "GenuineIntel"
```

See Also

_CPUID, _CPUIDD

{Created by Sjouke Hamstra; Last updated: 18/09/2014 by James Gaite}

_CPUIDD Function

Purpose

Returns processor information

Syntax

a = hex$(**_CPUIDD**)

Description

Every processor has an internal register containing information about its type and manufacturer. The information block is 128 bits (16 bytes) in size.

Example

Check Pentium type and if if MMX is supported.

```
OpenW 1
Print Btst(_CPUIDD, 15) // Pentium II or Pentium
  Pro by checking if "fcmove....." is available
// or
Print Btst(_CPUIDD, 23) // IsMMX
Do : Sleep : Until Me Is Nothing
```

Remarks:Since the Pentium III processor each has its own ID. The name and description of the _CPUIDD bits for Intel processors until Pentium III.

Bit	Name	Description
0	FPU	**Floating-point unit on-chip** - The processor contains an FPU that supports the

		Intel 387 floating-point instruction set. If Btst(_CPUIDD, 0) Then Print "FPU available"
1	VME	**Virtual Mode Extension** - The processor supports extensions to virtual-8086 mode.
2	DE	**Debugging Extension** - The processor supports I/O breakpoints, including the CR4.DE bit for enabling debug extensions and optional trapping of access to the DR4 and DR5 registers.
3	PSE	**Page Size Extension** - The processor supports 4-Mbyte pages.
4	TSC	**Time Stamp Counter** - The RDTSC instruction is supported including the CR4.TSD bit for access/privilege Control. If Btst(_CPUIDD, 4) Then _RDTSC possible.
5	MSR	**Model Specific Registers** - Model Specific Registers are implemented with the RDMSR, WRMSR instructions
6	PAE	**Physical Address Extension**
7	MCE	**Machine Check Exception**, Exception 18, and the CR4.MCE enable bit are supported
8	CX8	CMPXCHG8 Instruction Supported
9	APIC	On-chip APIC Hardware Supported
10		Reserved
11	SEP	Fast System Call Indicates whether the processor supports the Fast System Call instructions, SYSENTER and SYSEXIT. (Erratum in Pentium Pro, needs to examine _CPUID (Family 6, Model < 3, Stepping < 3: Not supported)
12	MTRR	**Memory Type Range Registers** supported (MTRR_CAP)

13	PGE	**Page Global Enable** - The global bit in the PDEs and PTEs and the CR4.PGE enable bit are supported.
14	MCA	**Machine Check Architecture** supported, specifically the MCG_CAP register.
15	CMOV	The processor supports CMOVcc, and if the FPU feature flag (bit 0) is also set, supports the FCMOVCC and FCOMI instructions. Pentium II+ and many Pentium Pro support somewhat faster Min and Max operations.
16	PAT	**Page Attribute Table** - Indicates whether the processor supports the Page Attribute Table. This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory on a 4K granularity through a linear address.
17	PSE-36	**36-bit Page Size Extension** - Indicates whether the processor supports 4-Mbyte pages that are capable of addressing physical memory beyond 4GB. This feature indicates that the upper four bits of the physical address of the 4-Mbyte page is encoded by bits 13-16 of the page directory entry.
18		Processor serial number is present and enabled. The processor supports the 96-bit processor serial number. feature, and the feature is enabled.
19		Reserved
20		Reserved
21		Reserved
22		Reserved
23		Intel Architecture MMX Technology supported

		If Btst(_CPUIDD, 23) or If IsMMX
24	FXSR	**Fast floating point save and restore** - Indicates whether the processor supports the FXSAVE and FXRSTOR instructions for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it uses the fast save/restore instructions.
25		Streaming SIMD Extensions supported (Pentium III+) (3D-Katmai command)
26		Reserved
27		Reserved
28		Reserved
29		Reserved
30		Reserved

The processor serial number for Pentium III processors can be obtained using **Btst**(_**CPUIDD**, 18) or _**CPUID** 3 in HEX in _**CPUID** and _**ECX** and _**EDX** (according Intel to show as 6 times 4 Hex characters in uppercase).

See the cpuid.g32 example

See Also

_CPUID$, _CPUID

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

IsMMX

Purpose

Returns a Boolean value indicating a whether the CPU supports MMX instructions.

Syntax

Bool = **IsMMX**

Description

Implemented for older CPUs (lower as Pentium 200). All newer CPUs support MMX.

Example

```
Print IsMMX
```

See Also

_CPUID

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

GetCurrentFiber Function

Purpose

Returns an identification value for the current running fiber.

Syntax

addr% = **GetCurrentFiber**()

Description

The return value is the address of the currently running fiber. A fiber is 'lightweight' thread, with less overhead and easier to maintain. A fiber uses less resources and the time to activate a fiber is lesser than for a thread.

The *CreateFiber* and *ConvertThreadToFiber* functions return the fiber address when the fiber is created. The **GetCurrentFiber** function allows you to retrieve the address at any other time.

The functions **GetTIB**, **GetCurrentFiber,** and **GetFiberData** are generated using inline code and are for this reason implemented in GFABASIC 32.

GetCurrentFiber . mov eax, fs:[16]

GetFiberData . mov eax, fs:[16] : . mov eax, [eax]

GetTIB . mov eax, fs:[24]

Example

Remarks

GetCurrrentFiber() and **GetFiberData**() are supported for Windows NT and later. These functions are used together with corresponding API functions like: *ConvertThreadToFiber*, *CreateFiber*, *SwitchToFiber*, etc..

See Also

GetFiberData, GetTIB

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

GetFiberData Function

Purpose

Reads the fiber data (data value) associated with the current fiber.

Syntax

x% = **GetFiberData**()

Description

The fiber data is the value passed to the *CreateFiber* or *ConvertThreadToFiber* API functions in the *lpParameter* parameter. This value is also received as the parameter to the fiber function. It is stored as part of the fiber state information.

This function is part of three fiber functions that are generated inline, **GetCurrentFiber**, **GetFiberData**, **and GetTIB**. These functions allow to connect to the base structure of the multitasking of Windows 95/98/NT/2000. For more information see **GetCurrentFiber**.

See Also

GetCurrentFiber, GetTIB

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

GetTIB Function

Purpose

Returns the linear address of a thread information block.

Syntax

x% = **GetTIB**()

Description

GetTIB() is generated as inline code and used to determine the linear address of a thread information block. A TIB contains the internal multitasking information of a thread. Because of the multitasking, a TIB cannot be stored in a Global (Public) or Static variable, only in a Local (register) variable is allowed. The structure of a TIB is almost undocumented, however the following variables of these block can be used.

{**GetTIB**()} is the address of the needed head of the list for the internal error handling. Used for Try/Catch/EndCatch in GFA-BASIC 32 respectively __try, __except/__finally in C.

{**GetTIB**() + 4} and {**GetTIB**() + 8} contain the maximum and minimum addresses of the stack.

{**GetTIB**() + 16} is **GetCurrentFiber**() and

{**GetTIB**() + 24} is a pointer to **GetTIB**() itself.

One possibility usage of **GetTIB**() is to check, if a program is running under a debugger. Under Windows 95/98

{**GetTIB**() + 32} is always zero if it runs under a debugger. To find out if Windows 95 or 98 is running use *GetVersion*(). Both highest bits of the return value are set, if it runs under Windows 95/98 (under Windows NT the highest bit is cleared).

Example

```
// This program was designed for OSs prior to
  WinMe & Win2000
If GetVersion() %& $c0000000 == $c0000000
  If {GetTIB() + 32}
    MsgBox "The program is running under a
      debugger"
  Else
    MsgBox "The program doesn't run not under a
      debugger"
  EndIf
Else
  MsgBox "The program doesn't run under Windows
    95/98"
EndIf
```

Remarks

Since Windows NT 4.0 and Windows 98 there exists a function named *IsDebuggerPresent*, which offers exact this functionality.

```
Declare Function IsDebuggerPresent Lib "kernel32"
  () As Int
Try
  If IsDebuggerPresent()
    MsgBox "The program is running under" _
      " a debugger"
  Else
    MsgBox "The program doesn't run under" _
```

```
      " a debugger"
  EndIf
Catch // for Windows 95 and NT 3.51
  If GetVersion() %& $c0000000 == $c0000000
    If {GetTIB() + 32}
      MsgBox "The program is running" _
        "under a debugger"
    Else
      MsgBox "The program doesn't run" _
        " under a debugger"
    EndIf
  Else
    MsgBox "Can't find the function" _
      " IsDebuggerPresent and the" _
      "program doesn't run under Windows 95"
  EndIf
End Catch
```

See Also

GetCurrentFiber, GetFiberData

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

^ Operator

Purpose

Raises a number to the power of another number.

Syntax

= x **^** y

x : iexp
y :avar

Description

The result is number raised to the power of exponent, always as a Double value. The value of exponent can be fractional, negative, or both.

Example

```
Global Int32 x, y
OpenW  1
x = 2
y = 8
Print x ^ y             // prints 256
Print (y - 2 * x) ^ 4   // prints 256
```

Remarks

When more than one exponentiation is performed in a single expression, the ^ operator is evaluated as it is encountered from left to right.

See Also

Pow, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 18/09/2014 by James Gaite}

* Operator

Purpose

Used to multiply two numbers.

Syntax

x * y

Description

The ***** operator is the arithmetic multiplication operator used to multiply an arithmetic expression.

Example

```
OpenW 1
Global x%, y%, a%
x% = 30
y% = 17
Print x * y    // Prints 510
KeyGet a%      // Press any key to close
CloseW 1
```

Remarks

When used with integers the compiler will optimize for integer math.

See Also

+, -, ^, *, /, \, %, Add, Sub, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

/ Operator, \ Operator

Purpose

Used to divide two numbers. The result is a double.

Syntax

# = exp1 **/** exp2

int = exp1 \ exp2

Description

The **/** operator always performs floating-point division. To force integer division use the \ operator or use **Div**.

Example

```
OpenW 1
Global x%, y%, a%
x = 36
y = 3
Print x / y  /* Result: 12
KeyGet a%
CloseW 1
```

See Also

+, -, ^, *, /F, \, %, Add, Sub, Mul, Div, ++, --, +=, -=, /=, *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

+ Operator

Purpose

Used to add/concatenate two expressions.

Syntax

r = **x + y**

Description

The **+** operator is the arithmetic addition operator when:

- Both expressions are numeric data types.
- One expression is numeric and the other is a **Variant** (except Null).
- Both **Variant** expressions are numeric.
- One **Variant** expression is numeric and the other is a string.

The **+** operator is a concatenation operator when:

- Both expressions are **String** data types.
- One expression is a **String** and the other is a **Variant** (except Null).
- Both **Variant** expressions are strings.

If either expression is **Null**, the result is Null.

For simple arithmetic addition involving only expressions of numeric data types, the data type of result is usually the same as that of the most precise expression. The order of precision, from least to most precise, is **Byte**, **Integer**,

Long, **Single**, **Double**, **Currency**, and **Large**. The following are exceptions to this order

- A **Single** added to a **Long** added results in a Double.
- A **Date** added to any data type results in a Date.

For **Variants** these exceptions apply:

- When the data type of result is a **Long**, **Single**, or **Date** variant that overflows its legal range, result is converted to a **Double** variant.

- When the data type of result is a **Byte** variant that overflows its legal range, result is converted to an **Integer** variant.

- When the data type of result is a **Short** variant that overflows its legal range, result is converted to a **Long** variant.

Example

```
OpenW 1
Global x% = 30, y# = 17 , a%
Print x + y
KeyGet a%
CloseW 1
```

Remarks

When used with integers the compiler will optimize for integer math.

When you use the + operator, you may not be able to determine whether addition or string concatenation will occur. Use the **&** or **$** operator for concatenation to eliminate ambiguity and provide self-documenting code.

See Also

$, & and +, -, ^, *, /, \, %, Dec, Inc, Add, Sub, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

- Operator

Purpose

Used to subtract numeric expressions or to indicate the negative value of a numeric expression.

Syntax 1

r = x **-** y

Syntax 2

- y

Description

The **-** operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the - operator is used as the unary negation operator (or sign) to indicate the negative value of an expression.

Example

```
Dim a% = 1, b! = 1
OpenW 1
Debug.Show : Debug.OnTop
Debug.Print a - b
Debug.Print a + -b
Do : Sleep : Until Win_1 Is Nothing
Debug.Hide
CloseW 1
```

Remarks

See Also

+, -, ^, *, /, \, %, Add, Sub, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

++ Command

Purpose

Increments a numeric variable.

Syntax

x++

x:numeric variable

Description

x ++ increments the value of x by 1.

Example

```
OpenW # 1
Dim x# = 2.7
x ++
Print x       // Prints 3.7
```

Remarks

Although ++ can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed.

Instead of x++ the following can be used instead:

```
x = x + 1
x := x + 1
x += 1
```

```
Inc x
Sub x, -1
Add x, 1
```

When integer variables are used ++ doesn't test for overflow!

See Also

+, -, ^, *, /, \, %, Add, Sub, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

-- Command

Purpose

Decrements a numeric variable.

Syntax

x--

x:avar

Description

x-- decrements the value of x by 1.

Example

```
OpenW # 1
Dim x# = 2.7
x--
Print x    // Prints 1.7
```

Remarks

Although -- can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed.

Instead of x--, the following can be used instead:

```
x = x - 1
x -= 1
Dec x
```

```
Sub x, 1
Add x, -1
```

NOTE: When integer variables are used, -- doesn't test for overflow!

See Also

+, -, ^, *, /, \, %, Add, Sub, Mul, Div, ++, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

+= Assignment

Purpose

Adds a numeric expression to a numeric variable.

Syntax

x **+=** y

x:variable
y:aexp

Description

x += y adds the expression y to the value in variable x.

Example

```
OpenW # 1
Dim x# = 17
x += 5 * 5
Print x     // Prints 42
```

Remarks

Although += can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed.

Instead of x += y the following can be used instead:

```
x = x + y
x := x + y
```

```
Add x, y
```

When integer variables are used += doesn't test for overflow!

See Also

+, -, ^, *, /, \, %, Dec, (dec,popfont,9,9,-1,-1)">Dec, Inc, Add, Sub, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

-= Assignment

Purpose

Subtracts a numeric expression from a numeric variable.

Syntax

x **-=** y

x:variable
y:aexp

Description

x -= y subtracts the expression y from the value in variable x.

Example

```
OpenW # 1
Dim x = 57
x -= 3 * 5
Print x    // Prints 42
```

Remarks

Although -= can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed.

Instead of x-=y

```
x = x - y
```

```
x := x - y
Sub x, y
```

can be used also. When integer variables are used -= doesn't test for overflow!

See Also

+, -, ^, *, /, \, %, Add, Sub, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

/= Assignment

Purpose

Divides the value of a variable or property by the value of an expression and assigns the result to the variable or property.

Syntax

x **/=** y

x:variable
y:aexp

Description

x **/=** y divides the expression y into the value in variable x. The type of the operation is determined by the data type of the variable x. For integer variables GFA-BASIC 32 generates integer division code.

Example

```
Local x% = 126
x% /= 2 + 1
Print x%    // Prints 42
```

Remarks

Although **/=** can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed.

See Also

+, -, ^, *, /F, \, %, Add, Sub, Mul, Div, ++, --, +=, -=, /=
, *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

*= Assignment

Purpose

Multiplies a numeric variable with a numeric expression.

Syntax

x ***=** y

x:numeric variable
y:aexp

Description

x *= y multiplies the value in variable x with the expression y.

Example

```
OpenW # 1
Dim x# = 6
x *= 9
Print x  // Prints 54
```

Remarks

Although *= can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed.

The following can be used instead of x*=y:

```
x = x * y
```

```
x := x * y
Mul x, y
```

When integer variables are used *= doesn't test for overflow!

See Also

+, -, ^, *, /, \, %, Add, Sub, Mul, Div, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 24/06/2017 by James Gaite}

%= Assignment Operator

Purpose

Take modulus of the first operand specified by the value of the second operand; store the result in the object specified by the first operand.

Syntax

i **%=** j
i : avar
j : avar

Description

Using this operator is almost the same as specifying *result = result % expression*, except that result is only evaluated once.

Example

```
Global l As Long
l = 42
l %= 5          // l = 42 Mod 5
Print l         // Prints  2
```

See Also

%, FMod, Mod, Mod(), Mod8, Mod8(), Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

&= Assignment Operator

Purpose

A logical bit-wise AND of two bit patterns, whereby the first pattern must be in an integer variable.

Syntax

i **&=** j

i: ivar

j: integer expression

Description

i **&=** j sets, in the integer variable i, only the bits which are set in both i and j.

Example

```
Print Bin$(3, 4)  // Prints 0011
Print Bin$(10, 4) // Prints 1010
Local i% = 3
i% &= 10
Print Bin$(i%, 4) // Prints 0010
```

Remarks

i = i **%&** j or i = i **And** j are synonymous with i **&=** j and can be used instead.

See Also

^=, |=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

|= Assignment

Purpose

Performs a logical bit-wise Or on two bit patterns, whereby the first pattern must be in an integer variable.

Syntax

i **|=** j

i:ivar

j:integer expression

Description

I **|=** j sets in the integer variable i, only the bits which are set in either i or j.

Example

```
Print Bin$(3, 4)  // Prints 0011
Print Bin$(10, 4) // Prints 1010
Local i% = 3
i% |= 10
Print Bin$(i%, 4) // Prints 1011
```

Remarks

i = i **|** j and i = i **Or** j are synonymous with i **|=** j and can be used instead.

See Also

&=, ^=

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

^= Assignment

Purpose

Performs an exclusive bit-wise OR on two bit patterns, whereby the first pattern must be in an integer variable.

Syntax

i **^=** j

i:ivar
j:integer expression

Description

i **^=** j sets in the integer variable i only the bits which are set in i but are clear in j and vice versa.

The type of the operation is determined by the data type of the variable x. For integer variables GFA-BASIC 32 generates integer code.

Example

```
Local i%
OpenW # 1
Print Bin$(3, 4)  // Prints 0011
Print Bin$(10, 4) // Prints 1010
i% = 3
i% ^= 10
Print Bin$(i%, 4) // Prints 1001
```

Remarks

i=i **Xor** j is synonymous with i **^=** j and can be used instead.

See Also

&=, |=, ^=, ^, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 18/09/2014 by James Gaite}

< Comparison-Operator

Purpose

Less than comparison operator.

Syntax

? = exp1 **<** exp2

Description

The result of a relational expression is **True** if the tested relationship is true and **False** if it is false.

Example

```
OpenW # 1
Global Int x , y
x = 10, y = 12
If x < y Then Print "True"
```

See Also

<, >, <>, ><, =<, <=, >=., =>, !=, =, ==, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

> Comparison Operator

Purpose

Greater than comparison operator.

Syntax

? = exp1 **>** exp2

Description

The result of a relational expression is **True** if the tested relationship is true and **False** if it is false.

Example

```
OpenW # 1
Global x , y
x = 17, y = 12
Print (x > y ? "True" : "False")
```

See Also

<, >, <>, ><, =<, <=, >=., =>, !=, =, ==, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

!=, <> and >< Inequality Operators

Purpose

An inequality operator returns false if its operands are equal, true otherwise.

Syntax

? = exp1 != exp2

? = exp1 <> exp2

? = exp1 >< exp2

exp1, exp2: aexp

Description

For primitive and value types, an inequality operator will return true if the values of its operands are different, false otherwise. For the String type, it compares the values of the strings and returns false if they are identical (see Mode Compare for more information on comparing strings).

Example

```
OpenW # 1
Global Int32 i = 32, b = 30
Print b != i                // Prints True
i = 30
Print b <> i                // Prints False
Print (2 <> 1) != (2 >< 2)  // Prints True
```

```
Do
  Sleep
Until Win_1 Is Nothing
```

Remarks

The **<>**, **><** and **!=** operators are synonymous, the former coming from classic BASIC, the latter from C.

See Also

<, >, =<, <=, >=, =>, =, ==, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

<=, =< Comparison-Operators

Purpose

Less than or equal to comparison operator.

Syntax

? = exp1 **<=** exp2

? = exp1 **=<** exp2

Description

The result of a relational expression is **True** if the tested relationship is true and **False** if it is false.

Example

```
OpenW # 1
Global Int x , y
x = 17, y = 12
Print (x =< y ? "True" : "False")
```

See Also

<, >, <>, ><, =<, <=, >=., =>, !=, =, ==, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

>=, => Comparison Operators

Purpose

Greater than or equal to operator.

Syntax

? = exp1 **>=** exp2

? = exp1 **=>** exp2

Description

The result of a relational expression is True if the tested relationship is true and False if it is false.

Example

```
OpenW # 1
Global x , y
x = 17, y = 12
Print (x >= y ? "True" : "False")
```

See Also

<, >, <>, ><, =<, <=, >=, =>, !=, =, ==, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

=, == Comparison Operators

Purpose

Equality comparison operators.

Syntax

? = exp1 **=** exp2

? = exp1 **==** exp2

Description

The result of a relational expression is **True** if both expression are equal, otherwise tested relationship is **False**.

Example

```
OpenW # 1
Global x , y
x = 17, y = 12
Print(x = y ? "True" : "False")
```

Remarks

As an alternative the C equality comparison operator **==** can be used.

See Also

<, >, <>, ><, =<, <=, >=., =>, !=, =, ==, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

&& Logical operator

Purpose

Logical AND of a true/false status of two or more values

Syntax

i **&&** j

i, j:arguments

Description

If you want to test whether two or more conditions are true, you can use the logical AND operator **&&**. This function is optimised for performance as it evaluates all conditions from left to right and if it comes across a false condition, any further conditions are not evaluated.

Example

```
ff("String", 150)   // Prints True
ff("", 150)         // Prints False
ff("String", 75)    // Prints False
ff("", 75)          // Prints False

Function ff(a$, height)
  Print Len(a$) && height => 100
  (* Prints True only if both conditions are True
    and/or non-zero *)
EndFunction
```

Remarks

The logical operator **And** can be used to perform a similar task but does not have the speed optimization of **&&** and can occasionally produce an odd result.

```
Local Byte a = 1, b = 2
Print a = 2 && b = 2  // Prints False (quick)
Print a = 2 And b = 2 // Prints 0 (slow)
```

This logical operator should not be confused with the bitwise AND operator **%&** which returns erroneous results if used in this way, as can be seen below:

```
Local Byte a = 1, b = 2
Print a = 1 && b = 2  // Prints True
Print a = 1 And b = 2 // Prints -1
Print a = 1 %& b = 2  // Prints False
```

See Also

If, ||, ^^, !, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

|| Logical operator

Purpose

Logical OR of a true/false status of two values

Syntax

i **||** j

i, j:function arguments

Description

To test whether either of two conditions is true (or if both are true), use GFA-BASIC 32's logical OR operator **||**. The condition is evaluated form left to right. To evaluate to true only one of the conditions must be true. When the first condition is true, the second isn't evaluated.

Example

```
Dim a% = 10
If a% = 0 || ff() Then Print "Both evaluated"
If a% || ff() Then Print "Only one evaluated"

Function ff() As Int
  Return 1
EndFunction
```

|| is a **logical** OR operator (and can be replaced by OR) but is not the same as the **bitwise** OR operators **|**, or **%|.** Replacing **||** with either of these would first evaluate both conditions, which are then bitwise Or-ed. Then the result of the bitwise or operation is tested for true or false, as shown below:

```
Dim a% = 10, b% = 2
Print a% = 10 Or b% = 5 // Prints -1 (True)
Print a% = 10 || b% = 5 // Prints True
Print a% = 10 %| b% = 5 // Prints False
Print a% = 10 | b% = 5  // Prints False
```

See Also

If, &&, ^^, !

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

^^ Logical operator

Purpose

Logical Xor of true/false status of two values

Syntax

? = exp1 **^^** exp2

Description

If you want to test whether one - and only one - of the expression is logically true (either-or), you can use the logical XOR operator **^^**. The condition is evaluated form left to right. To evaluate to true only one of the conditions must be true.

```
If Len(a$) ^^ height => 100
```

This expression evaluates to true when a$ contains data and the height variable is less than 100, or when a$ is empty and height is greater or equal to 100.

Example

```
Dim a% = 10, ff% = 5
Print "a% = 3 XOR ff = 5 " & (a% = 3 ^^ ff = 5 ?
  "is true" : "is false")    /* Returns True
Print "a% = 10 XOR ff = 5 " & (a% = 10 ^^ ff = 5 ?
  "is true" : "is false") /* Returns False
```

See Also

&&, ||, !, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 18/09/2014 by James Gaite}

! Logical Negation

Purpose

Logical negation of a Boolean value

Syntax

! i

i:function argument

Description

! i returns 0 when i is not zero and -1 when i equals 0.

Example

```
OpenW # 1
Local i#
i = 32
Print ! i     // Prints False
i = 0
Print ! i     // Prints True
```

See Also

&&,, ||, ^^, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

~ Command

Purpose

Voids a numeric expression.

Syntax

~a

Description

~ causes a calculated value or an integer expression returned from a function not to be put on stack or in a register. This means that the value is indeed calculated but because of ~ it's immediately "forgotten".

Example

This example performs a delay by calculating a complex expression which makes it very dependant on both computer and clock rate. It is much better to use Pause 1 here.

```
OpenW 1
Local i%, x%
For i% = 0 To _X - 1
  Plot i%, (SinQ (i%) + 1) * _Y / 2  // Plots the
    Sine Curve
  ~Sin(Cos(Tan(Log(2.3))))           // Is
    calculated but not used
Next
KeyGet x%
CloseW 1
```

The following example creates a **PopUp** menu called POP-UP Menu with entries L1, L2 and L3, without monitoring which entry was selected.

```
Local xo% = 100
Local yo% = 20
Local a$ = "POP-UP Menu | L _1 | L _2| L _3"
~PopUp(a$, xo%, yo%, 1)
```

Finally, this example produces a Message Box for which you do not need a return value.

```
~MsgBox("Press 'OK' to continue", MB_OK, "MsgBox")
```

Remarks

~x and **Void** x are equivalent to dummy% = x

New. ~ is also used to void a return value from a user defined function. However, in GFA-BASIC 32 the ~ is no longer needed.

```
Local d# = DoFunc(1)  // call function and store
  return value
~DoFunc(2)            // call function and void
  the return value
Print DoFunc(3)       // call function and print
  return value

Function DoFunc(a#)
  Return a# * 1.0
EndFunc
```

New. This is also true for DLL functions declared with **Declare**. The ~ is no longer necessary to void the return value. In addition, DLL functions are no longer called using

@@ or ^^, but simply by their name as if they were common functions.

However, ~ is still necessary for built-in API functions.

```
Declare Function GetUserName Lib "advapi32.dll"
  Alias "GetUserNameA" (ByVal lpBuffer As String,
  nSize As Long) As Long
OpenW 1
Dim n$ = String$( 30, #0)
GetUserName(n$, 30)                    ' New Syntax
Print "User Name: "; n$
~GetWindowText(Win_1.hWnd, V:n$, 30) ' Old Style:
  still uses ~
Print "Window Title: "; n$
```

See Also

Void

{Created by Sjouke Hamstra; Last updated: 20/09/2017 by James Gaite}

%& Operator

Purpose

Performs a logical bit-wise AND of two bit patterns.

Syntax

i **%&** j

i, j:integer expression

Description

i **%&** j sets in the result only the bits which are set in both i and j.

Example

```
Print Bin$(3 %& 7, 4)  // prints 0011
```

Remarks

And is synonymous with **%&** and can be used instead:

```
Print Bin$(3 And 7, 4)  // prints 0011
```

See Also

And, Or, Xor, Imp, Eqv, |, %|, ^^, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

|, %| Function

Purpose

Performs a logical bit-wise Or on two bit patterns.

Syntax

i **|** j

i **%|** j

i, j:integer expression

Description

i **|** j sets only the bits which are set in at least one of the two operands i or j. For completeness with **%&** (which replaces the **&** - And operator), the **%|** operator is added as a replacement for **|**.

Example

```
Print Bin$(3, 4)        // Prints 0011
Print Bin$(3 | 10, 4)   // Prints 1011
Print Bin$(5 %| 10, 4) // Prints 1111
```

Remarks

Or is synonymous with | and can be used instead:

```
Print Bin$(3 Or 10, 4) // Prints 1011
```

See Also

And, Or, Xor, Imp, Eqv, %&, ~

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

%| Operator

Purpose

Performs a logical bit-wise Or of two bit patterns.

Syntax

i **%|** j

i, j:integer expression

Description

I **%|** j sets in the result only the bits which are set in both i and j. The **%|** bitwise operator is equivalent as the 'older' **|** operator. The **%|** operator is provided in complement of the new **%&** bitwise and operator.

Example

```
Print Bin$(3 %| 2, 4)  // Prints 0011
```

Remarks

Or is synonymous with **%|** and can be used instead:

```
Print Bin$(3 Or 10, 4)  // Prints 1011
```

See Also

And, Or, Xor, Imp, Eqv, %&, ^^, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

%^ Operator

Purpose

Performs a bitwise exclusive OR operation between two integer values.

Syntax

i **%^** j

Description

The bitwise-exclusive-OR operator **%^** compares each bit of its first operand to the corresponding bit of its second operand. If one bit is 0 and the other bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

Bit 1	Bit 2	Result
0	0	0
1	0	1
0	1	1
1	1	0

%^ is equivalent to the **^** operator. For more information see %&.

Example

```
Print Bin$(3, 4)          // prints 0011
Print Bin$(3 %^ 10, 4)    // prints 1001
```

Remarks

x **Xor** y is synonymous with x **%^** y and can be used instead.

See Also

Xor, And, %&, Or(),%|, Imp(), Eqv(), Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

And Command, Function and Operator

Purpose

And can be used as a command, an operator, or as a function. It performs a logical bitwise And of two bit patterns

Syntax

And ivar, j(command)

int = i **And** j(operator)

int = **And**(i, j)(function)

ivar:integer variable

i, j:integer expression

Description

And ivar, j sets in the variable ivar the bits which are set in both ivar and value j.

i **And** j and(i, j) set in the result only the bits which are set in both i and j.

Example

```
Print Bin$(3, 4)            // Prints 0011
Print Bin$(10, 4)           // Prints 1010
Print Bin$(3 And 10, 4)     // Prints 0010
Print Bin$(And(3, 10), 4) // Prints 0010
Local a% = 3
And a%, 4
Print Bin$(a%, 4)         // Prints 0011
```

Remarks

The operator **And** is synonymous with **%&** and can be used instead:

```
Print Bin$(3 %& 10, 4) // Prints 0010
```

See Also

Or, Xor, Imp, Eqv, %&, |, ~, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Or, Or8 Functions

Purpose

It performs a logical bitwise OR of two bit patterns

Syntax

Or ivar, j(command)
int = i **Or** j(operator)
int = **Or**(i, j [,m,...])(function)

large =x **Or8** y(operator)
large = **Or8**(x, y [,z,...])(function)

ivar:integer variable
i, j:integer expression
x,y,z...:64-bit integer expression

Description

Or can be used as a command, an operator, and as a function, **Or8** only as an operator and function.

Or sets only the bits which are set in at least one of the two operands i or j.

Bit 1	**Bit 2**	**Result**
0	0	0
0	1	1
1	0	1
1	1	1

Example

```
Local a As Int32 = 3, b As Large = 2 ^ 45
Print a Or 6
Print b Or8 2 ^ 19
Print Or(a, 6)
Print Or8(b, 2 ^ 19)
Or a, 6 : Print a
Or b, 2 ^ 19 : Print b
```

Remarks

The operator **Or** is synonymous with **%|** (or **|**) and can be used instead:

```
Print Bin$(3 Or 10, 4) // Prints 1011
Print Bin$(3 %| 10, 4) // Prints 1011
Print Bin$(3 | 10, 4)  // Prints 1011
```

See Also

<Xor, Imp, Eqv, %&, |, ~, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Xor and Xor8 Functions

Purpose

Xor can be used as a command, an operator, and as a function, **Xor8** just as operator and function. They both perform an exclusive bit-wise Or on two bit patterns.

Syntax

Xor ivar, j(command)
int = i **Xor** j(operator)
int = **Xor**(i, j [,m,...])(function)

int64 = i **Xor8** j(operator)
int64 = **Xor8**(i, j)(function)

int:32-bit integer variable
int64:64-bit integer variable
i,j:integer expression

Description

i **Xor** j sets only the bits which are set in one - and only one - of the operands.

Bit 1	**Bit 2**	**Result**
0	0	0
0	1	1
1	0	1
1	1	0

The arguments are converted to **Long** (or **Large** for Xor8) before the operation is performed (using **CLong**).

Example

```
Debug.Show
Trace Bin$(3, 4)                    // Prints  0011
Trace Bin$(10, 4)                   // Prints  1010
Trace Bin$(Xor(3, 10), 4)           // Prints  1001
Local a% = 3
Xor a%, 4
Trace Bin$(a%, 4)                   // Prints  0111
Trace Bin$(Xor8(3, 10), 4)          // Prints  1001
```

Remarks

See Also

And ,Or, Xor, Imp, Eqv, %&, |, ~, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Imp and Imp8 Functions

Purpose

Used to perform a logical implication on two expressions.

Syntax

int = i **Imp** j(operator)
int = **Imp**(i, j [,m,...])(function)

int64 = i **Imp8** j(operator)
int64 = **Imp8**(i, j [,m,...])(function)

i, j, m:integer expression

Description

i **Imp** j combines the expressions i and j based on their order. The result is equivalent to a logical sequence. This means that something is false only when a true statement is followed by a false one. For expressions i and j this applies to their binary representation, i.e. the resulting bit will be 0 only when the corresponding bit in the first argument (i) is 1 and in the second argument (j) is 0.

Bit 1	Bit 2	Result
0	0	1
0	1	1
1	0	0
1	1	1

Imp8 should be used for 64-bit integers.

Example

```
OpenW # 1
Print 3 Imp 10 // Prints -2
```

3 **Imp** 10 returns -2. To understand this all 32 bits must be examined:

Bin$(3,32) = 00000000000000000000000000000011

Bin$(10,32) = 00000000000000000000000000001010

Bin$(3 Imp 10),32) =
11111111111111111111111111111110

The result of 3 **Imp** 10 is therefore -2.

Remarks

Imp is the only bit-wise operator for which the order of the arguments is important. This is because the result will produce a 0 only when, at the same position, a "true" (1) in the first argument is followed by a "false" (0) in the second argument.

This is why **Imp**(3,10) returns the value -2 (see above), but 10 **Imp** 3 returns -9:

Bin$(10,32) = 00000000000000000000000000001010

Bin$(3,32) = 00000000000000000000000000000011

Bin$((10 Imp 3),32)=
11111111111111111111111111110111

10 Imp 3 = -9

See Also

And ,Or, Xor, Imp, Eqv, %&, |, ~, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Eqv Function

Purpose

Returns the bit-wise equivalent of two bit patterns.

Syntax

int = i **Eqv** j(operator)
int = **Eqv**(i, j [,m,...])(function)

i, j, m:integer expression

Description

Eqv(i, j) sets in the result only the bits which are the same in both i and j. The arguments are converted to Integer before the operation is performed (using **CInt**).

Bit 1	**Bit 2**	**Result**
0	0	0
0	1	0
1	0	0
1	1	1

Example

```
Debug.Show
Trace Bin$(3, 4) // prints 0011
Trace Bin$(3 Eqv 10, 4)// prints 0110
Trace Bin$(Not (3 Xor 10), 4) // prints 0110
```

3 **Eqv** 10 returns the value -10. This result is easier to understand when all 32 bits are shown:

```
Debug.Show
Trace Bin$(3, 32)         //
  00000000000000000000000000000011
Trace Bin$(10, 32)        //
  00000000000000000000000000001010
Trace Bin$(3 Eqv 10, 32) //
  11111111111111111111111111110110
```

Remarks

This function is equivalent to **Not**(**Xor**(i, j)).

See Also

And ,Or, Xor, Imp, Eqv, %&, |, ~, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

And8 Operator and Function

Purpose

And8 can be used as an operator and as a function. It performs a logical bitwise And of two bit patterns and puts the result in a 64-bit integer.

Syntax

int64 = i **And8** j(operator)

int64 = **And8**(i, j)(function)

int64:64-bit integer variable
i, j:any numeric expression

Description

i **And8** j and **And8**(i, j) set in the result only the bits which are set in both i and j. Before the operation is applied, the arguments are converted to **Large** (using **CLarge**).

Example

```
Print Bin$(3 And8 10, 4)    // Prints 0010
Print Bin$(And8(3, 10), 4) // Prints 0010
```

See Also

Or8, Xor8, Eqv8, Imp8, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Eqv8 Function

Purpose

Returns the bit-wise equivalent of two 64-bit patterns.

Syntax

large = i **Eqv8** j(operator)
large = **Eqv8**(i, j [,m, ...])(function)

i, j, m:64-bit integer expression

Description

Eqv8(i, j) sets in the result only the bits which are the same in both i and j. The arguments are converted to Large before the operation is performed (using **CLarge**).

Bit 1	**Bit 2**	**Result**
0	0	0
0	1	0
1	0	0
1	1	1

Example

```
Debug.Show
Local a As Large, b As Currency, x%
a = Random(10000), b = Random(10000)
Trace Bin$(a, 64)
Trace Bin$(b, 64)
Trace Bin$(Xor8(a, b), 64)
Trace Bin$(Eqv8(a, b), 64)
Trace Bin$(Not (Xor8(a, b)), 64)
Local Large x, y, xx, a%
x = Large 12345678909
y = Large 10015432101
xx = x Eqv8 y
Trace xx
```

Remarks

This function is equivalent to **Not**(**Xor8**(i, j)).

See Also

And8, Or8, Xor8, Imp8, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

* AddressOf Operator

Purpose

Used to obtain the memory/descriptor address of a variable.

Syntax

% = ***var

var: variable

Description

The ***** operator is used as the address-of operator like **V:** (and **VarPtr**) and **ArrPtr**.

For variable-length strings and arrays ***** returns the address of the descriptor and behaves as **ArrPtr**(). For all other variables the ***** operator returns the memory address of data contained in that variable.

For a fixed string ***** returns the first four bytes of the data. This function has no meaning for a fixed-string.

Example

```
OpenW 1
Global x%, y$
Print *x%
KeyGet y$  // Press a key to end
CloseW 1
```

Remarks

The * operator is synonym to **ArrPtr** for strings and arrays, and **VarPtr**() and **V:** for other variables, fixed strings excluded.

See Also

ArrPtr, Varptr, V:

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

Operator Hierarchy

( )	**parenthesis**
+ - | !	unary plus, unary minus, bitwise NOT, logical NOT
$ &	explicit string addition
^	the power of
*** /**	multiply, divide (floating-point)
\ Div Mul	integer division, integer multiplication
% Mod Fmod	integer and the floating point modulo
+ - Add Sub	addition (the string addition, too) and subtraction
<< >> Shl Shr Rol Ror	all shift and rotate operators (also: Shl%, Rol|, Sar8, etc.)
%&	bitwise And
%| |	the bitwise Or
= == < > <= >= !=	all comparisons (also: NEAR ...)
And	bitwise And
Or	bitwise Or
Xor Imp Eqv	bitwise exclusive Or, implication and equivalence
&&	logical And
||	logical Or
^^	logical exclusive Or
Not	bitwise complement
? :	conditional expression
=, :=	assignment
***=, /=, %=, +=, -=, <<=,**	compound assignment

>>=, &=, ^=, |=

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

ChildW Command

Purpose

Creates a MDI Child window within a MDI parent window.

Syntax

ChildW [options] n, ph [,x, y, w, h, style]

ChildW [options] **Owner** form, n [,x, y, w, h, style]

ChildW [options] **Parent** form, n [,x, y, w, h, style]

n, ph, x ,y, w, h, style:integer expression
form:Form object name
options:[Tool] [Center] [Full] [Hidden] [Client3D] [Help] [Top] [Palette] [NoCaption] [NoTitle] [Fixed][Default] [MdiChild]

Description

ChildW creates the child window specified in *n* within the window specified in ph. The upper left corner coordinates of the Child window are given *x* and *y*, while the width and the height are given in *w* and *h*. The last parameter *style* is used to configure the window and can take WS_ window style constants. For an overview of the window styles see **ParentW**.

Alternative **ChildW** can take a Form object as parent using the syntax **Child Owner** *form* or **Child Parent** *form.*

ChildW creates a Form object named **Win**_n, where n is a number between 0 and 31. Although the GFA-BASIC 16 window management commands like **MoveW**, **SizeW**, etc. are still present, the window should be managed using the **Form** properties and methods. In the same tradition messages should be handled using event subs, like Win_1_Activate.

When **ChildW** specifies a number > 31, then the properties and methods are accessed using **Form**(n).property and the event sub are like Sub Form_Activate(Index%). The window number is passed as the first argument in the sub parameter list.

The *options* argument specifies additional window state settings.

Center	centers the form.
Full	creates a maximized window, excludes **Hidden** (full windows are always visible).
Hidden	invisible
Client3D	set WS_EX_CLIENTEDGE
Tool	creates a WS_EX_TOOLWINDOW
Help	includes a Help button in the window caption, excludes minimize an maximize buttons
Top	creates a topmost window
Palette	creates a WS_EX_PALETTEWINDOW
Fixed	a non-sizable window
NoCaption	no title bar
NoTitle	no title bar, alias
Default	uses Windows default values

Not all options are relevant for a MDI child window.

Example

```
ParentW 1
Local s% = WS_CAPTION | WS_OVERLAPPED | WS_VISIBLE
ChildW 2, 1, 20, 20, _X / 2, _Y / 2, s%
ChildW 44, 1
ChildW Owner Win_1, 3
Do
  Sleep
Until Win_1 Is Nothing
```

Opens a parent window (1) and within it two child windows (2 and 3). The position and size of the first child window are given while the second window (3) corresponds to the Windows position and size.

Remarks

In code MDI parent and child windows can also be created using the **Form** command.

```
Form MdiParent frmp
Form MdiChild Parent frmp, frmc
Form MdiChild Parent frmp, frmc1
Form MdiChild Parent frmp, frmc2
Form MdiChild Parent frmp, frmc3
//ChildW Owner frmp, 1
//ChildW Parent frmp, # 1
Set Me = frmc
Me.SetFocus
//frmp.MdiTile //Demo
Print frmp.Name
Print frmc.Name
Print "Child? ="; Me.MdiChild
Print "Parent? ="; Me.MdiParent
Print "Parent.Parent? ="; Me.Parent.MdiParent
Print "hMdiClientWnd ="; Me.Parent.hMdiClientWnd
```

```
Do
  Sleep
Loop Until Me Is Nothing
```

See Also

Form, Iconic?(), Parent(), Visible?(), Zoomed?(), ShowW, ParentW, OpenW

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

ParentW Command

Purpose

Creates a MDI parent window using API style flags to configure the window.

Syntax

ParentW [options] [#]n [, x, y, w, h][, style]

n, x, y, w, h, style:iexp
options:[Tool] [Center] [Full] [Hidden] [Client3D] [Help] [Top] [Palette] [NoCaption] [NoTitle] [Fixed][Default]

Description

ParentW n creates the window specified in n (between 0 and 31), whose upper left corner is given in x and y coordinates, the width in w and the height in h. The last parameter *style* is used to configure the window. *style* can take the following values which are "Or-ed":

WS_BORDER ($00800000)	window with a border
WS_CAPTION ($00C00000)	creates a window with a title. To make a system menu visible in such a window the WS_CAPTION and WS_POPUPWINDOW style elements must be combined.
WS_CLIPCHILDREN ($02000000)	clips all window output to the area outside of a child window.

WS_CLIPSIBLINGS ($04000000)	clips all window output within a child window to its client area.
WS_DISABLED ($08000000)	a window, which is initially inactive.
WS_DGLFRAME ($00400000)	a window with a double border but without a title.
WS_HSCROLL ($00100000)	a window with a horizontal scroll bar.
WS_ICONIC ($20000000)	a window which is initially displayed as an icon.
WS_MAXIMIZE ($01000000)	a window with maximum dimensions
WS_MAXIMIZEBOX ($00010000)	a window with a maximize box.
WS_MINIMIZE ($20000000)	a window with minimal dimensions.
WS_MINIMIZEBOX ($00020000)	a window with a minimize box.
WS_OVERLAPPED ($00000000)	an overlapping window. The window contains a border and a title. The client area overlaps with window border and title.
WS_OVERLAPPEDWINDOW (0xCF0000)	an overlapping window with following style elements: WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX
WS_POPUP ($80000000)	a popup window. Such window can't have the

	WS_CHILD attribute.
WS_POPUPWINDOW (0x80880000)	a popup window with following style elements: WS_BORDER | WS_POPUP | WS_SYSMENU
WS_SYSMENU ($00080000)	a window with a system menu in the title bar. Used only in windows with a title bar.
WS_TABSTOP ($00010000)	a window with a number of control elements which the user can arrive at by tapping the tab key. Used only in dialog boxes.
WS_THICKFRAME ($00040000)	a window with a thick border which is used to "size" the window.
WS_TILED (0x00000000)	
WS_VISIBLE ($10000000)	a window which is initially visible.
WS_VSCROLL ($00200000)	a window with a vertical scroll bar.

The **ParentW** command isn't the only way to create a parent MDI window in code. The alternative is to create a Form using the Form editor and setting **MdiParent** = True. At runtime **MdiParent** is read-only.

A MDI parent window can also be created using **Form MdiParent** or **OpenW MdiParent**, or by setting the **MdiParent** property in the Form Editor.

The *options* argument specifies additional window state settings.

Center - centers the form.

Full - creates a maximized window, excludes **Hidden** (full windows are always visible).

Hidden - opens invisible

Client3D - sets WS_EX_CLIENTEDGE

Tool - creates a WS_EX_TOOLWINDOW

Help - includes a Help button in the window caption, excludes minimize an maximize buttons

Top - creates a topmost window

Palette - creates a WS_EX_PALETTEWINDOW

Fixed - a non-sizable window

NoCaption - no title bar

NoTitle - no title bar, alias

Default - uses Windows default values

Example

```
' Ocx Form left aligned in a MDI parent window
' only way to create MDI parent in code:
ParentW 1, 20, 20, _X / 2, _Y / 2
Dim m$(80) : Local i%
For i = 0 To 60
  m(i) = i
Next
For i = 20 To 60 Step 20
  m(i) = ""
Next
```

```
m(61) = "&Window"
m(62) = "#1000#Cascade"
m(63) = "#1001#&Tile"
m(64) = "#1002#Tile 1"
m(65) = "#1003#Next"
m(66) = "#1004#&Previous"
Menu m()
Me.MenuItem(1004).Default = 1
Me.MdiSetMenu 3
Dim stpanel As Panel      ' create statusbar ocx
Ocx StatusBar stBar
.Panels.Add
Set stpanel = .Panels.Add : stpanel.AutoSize = 1
.Panels(1).ToolTipText = "Panel #1"
.Panels(2).ToolTipText = "Panel #2"
Ocx Form cld              ' create form ocx
.Width = 32 * Screen.TwipsPerPixelX
.Align = basLeft
.BackColor = RGB(192, 64, 64)
Me.ToolTipText = "ToolTip(ParentW)"
For i = 2 To 17           ' create mdi-child windows
  ChildW i, 1
  Me.Caption = Format(i, "'Window #'0")
  Me.ToolTipText = Format(i, "'This is a ToolTip
    for Window #'0")
Next
Do                        ' message loop
  Sleep
Loop Until Me Is Nothing

Sub Win_1_MenuOver(Idx%)
  'Trace "ov" & Idx
  If Idx < 0
    stBar.SimpleText = ""
  Else
    stBar.SimpleText = "MenuOver" & Idx
  End If
```

```
EndSub

Sub Win_1_MenuEvent(Idx%)
  'Trace "Ev" & Idx
  Switch Idx
  Case 1000 : Win_1.MdiCascade
  Case 1001 : Win_1.MdiTile
  Case 1002 : Win_1.MdiTile 1
  Case 1003 : Win_1.MdiNext
  Case 1004 : Win_1.MdiPrev
  Default : stpanel.Text = "MenuEvent" & Idx
  EndSwitch
End Sub

Sub cld_Paint
  Local i%
  Set Me = cld
  Color Me.ForeColor, Me.BackColor
  For i = 0 To 15
    Box 0, i * 16, _X, i * 16 + 16
    DrawText 1, i * 16, _X, i * 16 + 16, Format(i),
     _
     DT_CENTER | DT_VCENTER | DT_SINGLELINE
  Next
End Sub

Sub cld_MouseDown(Button&, Shift&, x!, y!)
  Local i%
  If Button == 1
    i = Int(y / 16)
    stpanel.Text = "red click" & i
  End If
End Sub

Sub cld_MouseMove(Button&, Shift&, x!, y!)
  Static Int lastI = -1
  Local i%
  i = Int(y / 16)
  If(i >= 16) i = 999
  If lastI != i
   lastI = i
   If i = 999
     cld.ToolTipText = "free"
   Else
     cld.ToolTipText = "red(Button)" & i
   End If
  End If
End Sub
```

Opens a parent window at position 20,20 with width _X/2 and height _Y/2, with a default style.

See Also

Form Object, Form, OpenW, ChildW

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

OcxOcx Command

Purpose

Creates an Ocx control with an Ocx parent in the current Form.

Syntax

OcxOcx parocx[(c_idx)] ocxtype name[(idx)] [[= text$] [,ID][, x, y, w, h] [, style]]

parocx:object variable name for the parent Ocx
c_idx:const iexp, control array index number
ocxtype:object typename
name:variable name (global)
idx:iexp, control array index number
text$:sexp, caption (optional)
ID:iexp, identifier value for the control
x, y, w, h:iexp, position and dimension of the object
style:iexp, additional windows style constants

Description

OcxOcx is used to create an Ocx control with some other Ocx control as its parent. OcxOcx is used in source code, rather than in the Form Editor. **OcxOcx** takes at least three arguments: an Ocx variable name, an Ocx type (OLE Control CoClass), and a variable name to which the new Ocx object is assigned.

parocx:object variable name for the parent Ocx
ocxtype:Ocx typename
name:Ocx variable name (global)

The *parocx* name represents the Ocx control that is to be the parent and *name* is the global variable name for the Ocx control in code. The parent Ocx *parocx* can be one of the following types:

parocx	***Meaning***
Form	A Form ocx can be used as a container (of course).
Image	A container with a small resource footprint. This could be used instead of a Form, which uses more resources (scaling, a DC, a Picture).
Frame	Particularly useful for **Option** Ocxes (RadioButtons). The **Transparent** property of the **Frame** may not be changed; otherwise, the embedded controls are invalid.
TabStrip	To embed (for instance) a Frame Ocx.
ToolBar	To embed (for instance) a ComboBox Ocx.
StatusBar	To embed (for instance) a Command Ocx

The *ocxtype* specifies the control to create. **OcxOcx** can be used to create all supported Ocx types: Animation, CheckBox, ComboBox, Command, CommDlg, Form, Frame, Image, ImageList, Label, ListBox, ListView, MonthView, Option, ProgressBar, RichEdit, Scroll, Slider, StatusBar, TabStrip, TextBox, Timer, ToolBar, TrayIcon, TreeView, UpDown.

The following statement creates a Button control (Ocx type is Command) at position 10, 10 and with width = 80 and height = 24 pixels in an (**Ocx**) **Form**.

```
OpenW 1
OcxOcx Win_1 Command cmd1 = "OK", 10, 10, 80, 24
```

```
.Default = True
.FontBold = True
Do : Sleep : Until Win_1 Is Nothing
```

After an **Ocx** or **OcxOcx** command has been executed, a hidden **With** command is active with the Ocx object just created. The **With** is valid to the next **With** or to the place a new Ocx is created.

Note: The **OcxOcx** command is also present in the context menu the Form Editor (right button click on the control).

Example

```
OpenW 1
' Load a bitmap
Ocx ImageList iml
.ListImages.Add , "I",
  CreatePicture(LoadIcon(Null, IDI_WARNING))
Ocx ToolBar tb
.ImageList = iml
// The first button is normal button, with picture
  "I"
Local btn As Button
Set btn = .Buttons.Add( , , , , "I")
Set btn = .Buttons.Add( ,"cb", , 4)
.Buttons("cb").Width = 100
OcxOcx tb ComboBox cb = , btn.Left, _
  btn.Top, btn.Width, btn.Height * 8
Local i%
For i = 0 To 99
  cb.AddItem Rnd, i
Next
Ocx ListBox lb = , 300, 0, 100, btn.Height * 8
For i = 0 To 99
  lb.AddItem Rnd, i
Next
```

```
lb.Top = 0        ' Move the ListBox vertical below
  the ToolBar
Do
  Sleep
Loop Until Me Is Nothing

Sub cb_Click
  Print "cb_Click"
  Print cb.ItemData(cb.ListIndex)
  Print cb.Text
End Sub

Sub tb_Click
  Print "tb_Click"
End Sub

Sub lb_Click
  Print "lb_Click"
  Print lb.ItemData(lb.ListIndex)
  Print lb.Text
End Sub
```

Example – Using a control array.

```
OpenW 1
Const id_frame = 400    ' MUST BE A CONST!
Ocx Frame fra(id_frame) = "Colors" , 110, 40, 156,
  164
Local Int idx = id_frame + 1
OcxOcx fra(id_frame) Option opt(idx) = "Border",
  8, 020, 60, 20 : Inc idx
OcxOcx fra(id_frame) Option opt(idx) = "Label", 8,
  040, 60, 20 : Inc idx
OcxOcx fra(id_frame) Option opt(idx) = "Fore", 8,
  060, 60, 20 : Inc idx
Do
  Sleep
```

```
Until Me Is Nothing
```

Remarks

When using the control array syntax for the OcxOcx parent, the index must be of type **Const Int**, see the example above.

Ocx creates a control whose parent is **Me**. Therefore, **Ocx** is the same as **OcxOcx Me**.

```
OpenW 1 : TitleW 1, " Win 1"
OpenW 2 : TitleW 2, " Win 2"
' create a button in Win 1:
Set Me = Win_1 :
OcxOcx Me Command cmd1 = "GFA", 10, 10, 50, 20
' Button in current active window (Me)
Ocx Command cmd2 = "GFA2", 10, 40, 50, 20
Do
  Sleep
Loop Until Win_1 Is Nothing
CloseW 2
```

See Also

Ocx, OCX(), Me, Form, Command, Option, CheckBox, RichEdit, ImageList, TreeView, ListView, Timer, Slider, Scroll, Image, Label, ProgressBar, TextBox, StatusBar, ListBox, ComboBox, Frame, CommDlg, MonthView, TabStrip, TrayIcon, Animation, UpDown

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Form Function

Purpose

Returns a **Form** object for a given window handle or window number.

Syntax

Set form = **Form**(wh%)

form:Form Object
wh%:iexp

Description

Form(*handle***)** returns a **Form** object for a given window handle. When the handle can't be found the return value is **Nothing.** This type of function **Form()** is in particular useful in the event subs _**MessageProc** and **Screen_KeyPreview**.

Form(*n*) designates the window-form created with **OpenW** or **ChildW**. Where n is a number between 0 and 31, the name of froms is predefined as **Win_*n***, where n is a number between 0 and 31. This name is introduced in the global variable list and is accessible throughout the program. These variable names can be used in accessing properties, methods, and events. For instance, **Win_1.Name** returns "Win_1". Windows created with a number greater than 31 don't declare global variable names implicitly and can only be accessed using **Form**(n).Name. However, there is no variable name introduced, but their name still consists of "Win_n", where n is the window number.

Example 1:

```
Dim frm As Form
OpenW 1
Set frm = Form(Win_1.hWnd)
frm.Caption = "Window 1"
Form(1).BackColor = $8000000f
Win_1.FontTransparent = True
Print frm.Name
Do : Sleep : Until Win_1 Is Nothing
// Using frm is the Do...Until statement will
  cause the program into an infinite loop
// ...as closing Win_1 does not set frm to Nothing
  but does delete the object so
// ...that frm can no longer be accessed.
Trace IsNothing(frm)
Set frm = Nothing
```

Example 2:

```
Global key$, i%
Form test = , 0, 0, _X / 2, _Y / 2
For i% = 1 To 15 : OpenW i% : Next
PrintScroll = True
Do
  Try
    Print Form(GetActiveWindow()).Name
  Catch
  EndCatch
  Sleep
Until key$ <> ""
For i% = 1 To 15 : CloseW i% : Next
test.Close
```

```
Sub Screen_KeyPreview(hWnd%, uMsg%, wParam%,
  lParam%, Cancel?)
  If uMsg% = WM_CHAR Then key$ = wParam%
EndSub
```

Remarks

The **OCX**(*handle*) function does the same as **Form**(*Handle*) for an Ocx control by returning a general **Control** object for a given window handle.

See Also

OCX(), Ocx, OcxOcx, Form, Screen_KeyPreview

{Created by Sjouke Hamstra; Last updated: 15/07/2015 by James Gaite}

OCX() Function

Purpose

Returns a **Control** object for a given window handle.

Syntax

Set co = **OCX**(hWnd)

co:Control Object
hWnd:Handle

Description

OCX() returns the general Ocx **Control** object for a given window handle. When the handle can't be found the return value is **Nothing.** The exact type of the Ocx can be obtained using the **TypeOf**(co) or **TypeName**() function. An alternative is to check for the name of the control using the **Name** property.

The function **OCX()** is in particular useful in the event subs _MessageProc and **Screen_KeyPreview**.

Example

```
Form frm1 = "GFA-Test", 10, 10, 250, 170
Ocx Command cmd1 = "But_1", 30, 100, 45, 25
Ocx Command cmd2 = "But_2", 80, 100, 45, 25
Ocx Command cmd3 = "But_3", 130, 100, 45, 25
Local ho As Int,  co As Control, h As Int
Do
  Sleep
 h = GetFocus()
 If h <> ho
   ho = h
   Set co = OCX(h)
   If ! IsNothing(co)
     Print co.name
   EndIf
 EndIf
Loop Until Me Is Nothing
CloseW 1
```

Remarks

The **Form()** function does the same for a form window; it returns a **Form** object for a given window handle.

It is not possible to destroy an object created using **OCX**(); to get around this problem, simply set the **Width** property to 0 (zero).

See Also

Ocx, OcxOcx, Form, Form()

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Me Variable

Implicit declared Form variable containing the currently active form.

Syntax

Set Me = frm

Set frm = **Me**

Me.[property | method]

frm:Form object

Description

Me, at all times, holds the currently active Form object. When all forms are closed **Me** = **Nothing**. Me is often used in the main message loop to test for a valid Form object. As long as Me contains a valid object messages should be processed and the loop must continue.

Example

```
OpenW 1
Print "Me = "; Me.Name
OpenW 2
Print "Me = "; Me.Name
Do
  Sleep
Loop Until IsNothing(Me)
```

In GFA-BASIC 32 the message loop must be inserted explicitly. The **Sleep** command is responsible for retrieving the messages from the queue and for dispatching them to the Forms and OCX controls. The example above shows a minimal application.

See Also

Form, LoadForm, OpenW, Sleep, Ocx

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

Output Command

Purpose

Redirects the output to a **Form** or **Printer** object.

Syntax

Output = object

object:Form or Printer

Description

With **Output** the output from GFA-BASIC 32 graphic commands is redirected. The output can be redirected to a Printer object or to a Form object. The output can also be temporarily redirected to a Form which isn't currently active, for instance, to draw in the client area of a non-active window.

Example

```
Dim bmp As Picture
// please choose and set a file with path
Local d$ = Left(ProgName$, RInStr(ProgName$, "\"))
  & "gfawintb.bmp", h As Handle
Set bmp = LoadPicture(d$)
// to choose a printer and switch output
Dlg Print Me, 0, h
If h <> 0
  SetPrinterHDC h
  Output = Printer
  Printer.StartDoc "Test"
  Printer.StartPage
  Printer.PaintPicture bmp, 0, 0
  Printer.EndPage
  Printer.EndDoc
EndIf
```

Remarks

Set Me returns the output to a **Form**.

See Also

Me, Form, Printer

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

MouseCursor Property

Purpose

Returns or sets a value indicating the type of mouse pointer displayed when the mouse is over a particular part of an object at run-time.

Syntax

Object**.MouseCursor** [= CValue]

Object:Ocx Object
cvalue:MouseCursor Object

Description

The **MouseCursor** property takes a **MouseCursor** object, which is returned by **LoadCursor** for instance. The **MouseCursor** property provides a custom icon that is used when the **MousePointer** property is set to 98 (**basCursor**).

The **MouseCursor** object only has one property.

Handle - getHandle - Returns the hCursor handle of the cursor.

Example

```
OpenW 1
Local mc As MouseCursor
If Exist(WinDir & "\Cursors\hourglas.ani") // Only
 included up to WinXP
```

```
  Set mc = LoadCursor(WinDir &
    "\Cursors\hourglas.Ani")
Else
  Set mc = LoadCursor(WinDir &
    "\Cursors\aero_busy.ani")
EndIf
Set Win_1.MouseCursor = mc
Win_1.MousePointer = 98      // basCursor
Print Win_1.MouseCursor.Handle
Do
  Sleep
Until IsNothing(Me)
Set mc = Nothing
```

Remarks

See Also

LoadCursor, MouseIcon, MousePointer, DefMouse

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

LoadCursor Function

Purpose

Creates a **MouseCursor** object based on data contained in a file. The file is specified by its name and containing cursor data in either cursor (.CUR) or animated cursor (.ANI) format.

Syntax

Set mc = **LoadCursor**(file$)

mc:MouseCursor object
file$sexp

Description

LoadCursor loads a cursor file either from disk or from the ':Files' section. The return value is a MouseCursor object that can be assigned to MouseCursor properties of Ocx objects (for instance Form.MouseCursor). The **MouseCursor** object is activated when the **MousePointer** property of the Ocx object is set to **basCursor** (98).

Example

```
OpenW 1
Local mc As MouseCursor
If Exist(WinDir & "\Cursors\hourglas.ani") // Only
  included up to WinXP
  Set mc = LoadCursor(WinDir &
    "\Cursors\hourglas.Ani")
Else
```

```
  Set mc = LoadCursor(WinDir &
   "\Cursors\aero_busy.ani")
EndIf
Set Win_1.MouseCursor = mc
Win_1.MousePointer = 98      // basCursor
Do
  Sleep
Until IsNothing(Me)
Set mc = Nothing
```

Remarks

Since GFA-BASIC 32 uses the **LoadCursor** as a reserved name, the API function *LoadCursor*() has been renamed to **LoadResCursor** or **apiLoadCursor.**

See Also

MouseCursor

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

MouseIcon Property

Purpose

Returns or sets a custom mouse icon.

Syntax

Object.**MouseIcon** [= picture]

Object:Ocx Object
value:Picture Object

Description

The **MouseIcon** property provides a custom icon that is used when the **MousePointer** property is set to 99.

The **MouseIcon** property provides your program with easy access to custom cursors of any size, with any desired hot spot location. Visual Basic does not load animated cursor (.ani) files, even though 32-bit versions of Windows support these cursors.

Example

```
OpenW 1
Local mc As Picture
Set mc = CreatePicture(LoadIcon(Null,
  IDI_WARNING))
Set Win_1.MouseIcon = mc
Win_1.MousePointer = 99      // basIcon
Do
  Sleep
```

```
Until IsNothing(Me)
Set mc = Nothing
```

Remarks

See Also

MouseCursor, MouseIcon, MousePointer, DefMouse

{Created by Sjouke Hamstra; Last updated: 19/10/2014 by James Gaite}

Buttons, Button Objects

Purpose

A **Buttons** object is a collection of **Button** objects. A **Button** object represents an individual button in the **Buttons** collection of a **Toolbar** control.

Syntax

ToolBar.**Buttons**

ToolBar.**Buttons**(index)

ToolBar.**Button**(index)

index:Variant

Description

The *ToolBar*.**Buttons** property returns a reference to the **Buttons** object, a collection of **Button** objects.

ToolBar.**Buttons**(index) or **Button**(index) returns a reference to the **Button** with the given index (integer or string).

For each **Button** object, you can add text or a bitmap image, or both, from an **ImageList** control, and set properties to change its state and style. You can manipulate **Button** objects using standard collection methods (for example, the **Add** and **Remove** methods). Each element in the collection can be accessed by its index, the value of the

Index property, or by a unique key, the value of the **Key** property.

The **Buttons** properties and methods:

Add | Clear | Count | Item | Remove

The **Button** properties and methods:

Caption | Checked | Enabled | Height | Hidden | Image | Indeterminate | Index | Key | Left | Pressed | Style | Tag | ToolTipText | Top | Value | Width

Example

```
Ocx ToolBar tlb
tlb.Add , "open" , "Open"
tlb.Add , "save" , "Save"
Debug.Show
Trace tlb.Button(1).Caption
Trace tlb.Buttons(2).Key
Trace tlb("open").Index
Do : Sleep : Until Me Is Nothing
```

Known Issues

1. The **Toolbar** and **Buttons** methods **Clear** and **Remove** don't work correctly and will eventually crash GFA-BASIC 32.
2. Although the text of the **Button** caption can be retrieved using **Button.Caption**, the ability to set the caption after it has been created has never been implemented in GB32. There is no known workaround to this.

See Also

ToolBar

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

ColumnHeader, ColumnHeaders Objects

Purpose

A **ColumnHeaders** object is a collection of **ColumnHeader** objects.

A **ColumnHeader** object represents an individual column header in the **ColumnHeaders** collection of a **ListView** control.

Syntax

ListView.**ColumnHeaders**

ListView.**ColumnHeaders**[**.Item**](index)]

index:Variant

Description

The syntax above refers to the collection and to individual elements in the collection, respectively, according to the standard collection syntax.

The *ListView*.**ColumnHeaders** property returns a reference to the **ColumnHeaders** object, a collection of **ColumnHeader** objects.

ListView.**ColumnHeaders.Item**(index) returns a reference to the **ColumnHeader** with the given index (integer or string). Since **Item** is the default property it can be left out.

For each **ColumnHeader** object, you can add text, and set properties to change its alignment and width. You can manipulate **ColumnHeader** objects using standard collection methods (for example, the **Add** and **Remove** methods). Each element in the collection can be accessed by its index, the value of the **Index** property, or by a unique key, the value of the **Key** property.

The **ColumnHeaders** properties and methods:

Add | Clear | Count | Item | Remove

The **ColumnHeader** properties and methods:

Alignment | Index | Key | Left | ListViewName | SubItemIndex | Tag | Text | Width

Example

```
Dim ch As ColumnHeader
Ocx ListView lv1 = "", 10, 10, 400, 200 : .View =
  3
Set ch = lv1.ColumnHeaders.Add( , , "Column1") :
  ch.Width = TextWidth("  Column1  ") *
  Screen.TwipsPerPixelX
Set ch = lv1.ColumnHeaders.Add( , , "Column2") :
  ch.Width = TextWidth("  Column2  ") *
  Screen.TwipsPerPixelX
Set ch = lv1.ColumnHeaders.Add( , , "Column3") :
  ch.Alignment = 2
Set ch = lv1.ColumnHeaders.Add( , , "Column4") :
  ch.Alignment = 1
Do : Sleep : Until Me Is Nothing
```

See Also

ListView

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Panel, Panels Objects

Purpose

A **Panels** object is a collection of **Panel** objects. A **Panel** object represents an individual panel in the **Panels** collection of a **StatusBar** control.

Syntax

StatusBar.**Panels**

StatusBar.**Panels.Panel**(index)

StatusBar.**Panel**(index)

index:Variant

Description

Use the **Panels** collection to retrieve, add, or remove an individual **Panel** object.

The *StatusBar*.**Panels** property returns a reference to the **Panels** object, a collection of **Panel** objects.

StatusBar.**Panels.Panel**(index) returns a reference to the **Panel** with the given index (integer or string).

StatusBar.**Panel**(index) is a shortcut for the above.

A **Panel** object can contain text and a bitmap which may be used to reflect the status of an application.

To change the look of a panel, change the properties of the **Panel** object. To modify the properties at run-time, you can change the **Panel** object properties in code.

The **Panels** properties and methods:

Add | Clear | Count | Item | Remove

The **Panel** properties and methods:

Alignment | AutoSize | Index | Key | Left | MinWidth | Text | Style | Tag | ToolTipText | Top | Value | Width

Panel only properties

AutoSize returns or sets a value that determines the width of a **Panel** object after the **StatusBar** control has been resized.

0	(Default) None. No autosizing occurs. The width of the Panel is always and exactly that specified by the Width property.
1	Spring. When the parent form resizes and there is extra space available, all panels with this setting divide the space and grow accordingly. However, the panels' width never falls below that specified by the **MinWidth** property.
2	Content. The Panel is resized to fit its contents, however, the width will never fall below the width specified by the **MinWidth** property. **Panel** objects with the Content style have precedence over those with the Spring style. This means that a Spring-style **Panel** is shortened if a **Panel** with the Contents style requires that space.

MinWidth returns or sets the minimum width of a **StatusBar** control's **Panel** object. The **MinWidth** property is used when the **AutoSize** property is set to Contents or Spring, to prevent the panel from autosizing to a width that is too small. When the **AutoSize** property is set to 0, the **MinWidth** property is always set to the same value as the **Width** property.

The default value is the same as the default of the **Width** property. The *value* argument uses the same scale units as the scale mode of the parent form.

Example

```
Global Enum sbrNoAutoSize = 0, sbrSpring,
  sbrContents
Ocx StatusBar sb
sb.Panels.Add , , "Hello" : sb.Panel(1).AutoSize =
  sbrNoAutoSize
sb.Add , , "Hello" : sb.Panel(2).AutoSize =
  sbrSpring
sb.Panels.Add , , "Hello" : sb.Panel(3).AutoSize =
  sbrContents
sb.Add , , "Hello" : sb(4).MinWidth = 50 :
  sb(4).AutoSize = sbrContents
Do : Sleep  : Until Me Is Nothing
```

See Also

StatusBar

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

ListItem, ListItems Objects

Purpose

A **ListItem** consists of text, the index of an associated icon (**ListImage** object), and, in Report view, an array of strings representing subitems.

A **ListItems** object is a collection of **ListItem** objects.

Syntax

ListView.**ListItems**

ListView.**ListItems**[**.Item**](index)

index:Variant

Description

The syntax above refers to the collection and to individual elements in the collection, respectively, according to the standard collection syntax.

The *ListView*.**ListItems** property returns a reference to the **ListItems** object, a collection of **ListItem** objects.

ListView.**ListItems.Item**(index) returns a reference to the **ListItem** with the given index (integer or string). Since **Item** is the default property it can be left out.

For each **ListItem** object, you can add text and pictures. However, to use pictures, you must reference an **ImageList** control using the **Icons** and **SmallIcons** properties of the **ListView** Ocx.

You can also change the image by using the **Icon** or **SmallIcon** properties of the **ListItem** object.

You can manipulate **ListItem** objects using standard collection methods (for example, the **Add** and **Remove** methods). Each element in the collection can be accessed by its index, the value of the **Index** property, or by a unique key, the value of the **Key** property.

The **ListItems** collection properties and methods:

Add | Clear | Count | Item | Remove

The **ListItem** properties and methods:

AllText | BackColor | Bold | Checked | CreateDragImage | EnsureVisible | ForeColor | Ghosted | Icon | Index | Italic | Key | ListViewName | Selected | SmallIcon | SubItems | Tag | Text | Underline | Visible

The **CreateDragImage** is not implemented

Example

```
Dim li As ListItem
Ocx ListView lv = "", 10, 10, 300, 300 : .View = 3
  : .FullRowSelect = True
lv.ColumnHeaders.Add , , "Column1" :
  lv.ColumnHeaders.Add , , "Column 2"
Local n : For n = 1 To 26 : lv.Add , , ""
  Set li = lv.ListItems(n)
  li.AllText = "Item" & n & ";" & Chr(64 + n)
  If Odd(n) Then li.Bold = True
  If n / 3 = Int(n / 3) Then li.ForeColor =
    RGB(255, 0, 0)
  If n / 4 = Int(n / 4) Then li.BackColor =
    RGB(192, 192, 192)
```

```
Next n
Do : Sleep  : Until Me Is Nothing

Sub lv_Click
  If lv.SelectedCount <> 0
    Set li = lv.SelectedItem
    Message "Selected Row has the following
      items:"#13#10 & li.SubItems(0) & #13#10 &
      li.SubItems(1)
  EndIf
EndSub
```

Remarks

See Also

ListView

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Node, Nodes Objects (TreeView)

Purpose

A **Node** object is an item in a **TreeView** control that can contain images and text.

A **Nodes** object is a collection of **Node** objects.

Syntax

TreeView.**Nodes**

TreeView.**Nodes**[**.Item**](index)

index : Variant

Description

The syntax above refers to the collection and to individual elements in the collection, respectively, according to the standard collection syntax.

The *TreeView*.**Nodes** property returns a reference to the **Nodes** object, a collection of **Node** objects.

TreeView.**Nodes.Item**(index) returns a reference to the **Node** with the given index (integer or string). Since **Item** is the default property it can be left out.

For each **Node** object, you can add text and pictures. However, to use pictures, you must reference an **ImageList** control using the **ImageList** property of the **TreeView** Ocx.

Pictures can change depending on the state of the node; for example, a selected node can have a different picture from an unselected node if you set the **SelectedImage** property to an image from the associated **ImageList**.

You can manipulate **Node** objects using standard collection methods (for example, the **Add** and **Remove** methods). Each element in the collection can be accessed by its index, the value of the **Index** property, or by a unique key, the value of the **Key** property.

The **Nodes** collection properties and methods:

Add | AddFirst | AddLast | AddNext | AddPrev |AddChild | Clear | Count | Item | Remove

The **Node** properties and methods:

BackColor | Bold | Child | Children | CreateDragImage | EnsureVisible | Expanded | ExpandedImage | FirstSibling | ForeColor | FullPath | Index | Image | Italic | Key | LastSibling | Next | Parent | Previous | Root | Selected | SelectedImage | Sorted | Tag | Text | TreeViewName | Underline | Visible

Node only properties:

% = Children	**Returns the number of child Node objects contained in a Node object.**
CreateDragImage	Not implemented

Example

```
Dim node As Node
Ocx TreeView tv = "", 250, 10, 230, 200
tv.Add , , , "Painters"
tv.Nodes.Add  1, tvwChild , , "Da Vinci"
tv.Add 1, tvwChild, , "Titian"
tv.AddItem 1, tvwChild, , "Rembrandt"
Set node = tv.Nodes.Add(1, tvwChild, , "Goya")
Set node = tv.Add(1, tvwChild, "David" , "David")
tv.LineStyle = tvwRootLines
tv.Style = tvwTreelinesText
tv.Indentation = 25
tv("David").Italic = True
tv.Node(3).Bold = True
tv.Nodes(4).Underline = True
tv!David.EnsureVisible ' Expand tree to see all
  nodes.
tv.SetFocus
tv("David").Selected = 1
Do
  Sleep
Until Me Is Nothing
```

Remarks

GFA-BASIC 32 specific

Instead of explicitly using the **Nodes** collection to access a **Node** element, you can use a shorter notation. First, the **TreeView** supports an **Item** property:

tv.**Item**(idx)tv.**Nodes**.**Item**(idx)

Like the **Item** method of tv.**Nodes, Item** is the default method of **TreeView**. Therefore, a **Node** can be accessed as follows:

tv(idx)	tv.**Nodes**(idx)
tv!idx	tv.**Nodes**!idx

Each dot saves about 30 bytes of code.

To enumerate over the **Nodes** collection of a **TreeView** Ocx, use For Each on the Ocx control directly, like:

```
Local nod As Node
For Each nod In tv : DoSomething(nod) : Next
```

See Also

ListView

{Created by Sjouke Hamstra; Last updated: 22/10/2017 by James Gaite}

ListImages Collection, ListImage Object

Purpose

A **ListImages** collection is a collection of **ListImage** objects. A **ListImage** object is a bitmap of any size (**Picture** object).

Syntax

imagelist.**ListImages**

imagelist.**ListImages.ListImage(***index***)**

imagelist.**ListImages(***index***)**

imagelist.**ListImage(***index***)**

index:Variant

Description

The **ListImages** and **ListImage** are a property of the **ImageList** Ocx control. The **ListImages** collection holds all the images, wrapped in a **ListImage** object, for the **ImageList** control.

The **ListImages** collection is a 1-based collection. *index* is an integer or string that uniquely identifies the object in the collection. The integer is the value of the **Index** property; the string is the value of the **Key** property.

You can add and remove a **ListImage** at design time using the 'ImageList Data' dialog box of the **ImageList** Properties, or at run time using the **Add** method for **ListImage** objects.

ListImages Properties and Methods

Add | Clear | Count | Item | Remove

ListImage Properties and Methods

Draw, | ExtractIcon | Index | Key | Picture | Tag

Note The **ImageList** control is an Ocx object for a **ListImages** collection of **ListImage** objects, where **ListImage** object is a holder of a **Picture** object. Therefore, the **ImageList** Ocx control holds a collection of **Picture** objects.

Example

```
OpenW Full 1
Dim pic As Picture
Local Int32 n, p1
// Find picture file
Local d$ =
  GetSetting("\\HKEY_CLASSES_ROOT\Applications\GfaW
  in32.exe\shell\open\command", , "")
If Left(d$, 1) = #34 Then d$ = Mid(d$, 2)
n = RInStr(d$, "\") : If n <> 0 Then d$ = Left(d$,
  n - 1)
If Not Exist(d$ & "\gfawintb.bmp") Then _
  MsgBox("Can not locate gfawintb.bmp
    file"#13#10#13#10"Please manually place it in
    the GFABASIC32\Bin folder and try again.")  :
    End
Set pic = LoadPicture(d$ & "\gfawintb.bmp")
```

```
// Create ImageList and split picture up into
  separate icons
Ocx ImageList iml : .ImageHeight = 16 :
  .ImageWidth = 16
For n = 0 To 21 : iml.AddPart , , pic, (n * 16), 1
  : Next n
// Add icons to TreeView object
Ocx TreeView tv = "", 10, 10, 100, 400 :
  .ImageList = iml
For n = 1 To 22 : tv.Add  , , , "Icon" & n, n :
  Next n
// Draw the icons as pictures using PaintPicture
  (reproduces exact size of 16x16)
For n = 0 To 21 : Set pic = iml(n +
  1).ExtractIcon  : PaintPicture pic, 130, (n *
  18)  : Next n
// Draw the icons as pictures using DrawIcon
  (reproduces enlarged size of 32x32)
For n = 0 To 21 : Set pic = iml(n + 1).ExtractIcon
  : ~DrawIcon(Me.hDC, 160, (n * 34), pic.Handle) :
  Next n
Do : Sleep : Until Me Is Nothing
```

Remarks

Images can also be inserted by using the **ImageList** Control methods **Add** and **AddItem**. This is a bit shorter, both in code and in executable instructions.

```
Dim img As ListImage
iml.Add , "open", LoadPicture(":open")
iml.AddItem , "save", LoadPicture(":save")
Set img = iml.Add( , "print1",
  LoadPicture(":print1"))
```

In the same way, items can be obtained in a shorter way. Use the **ImageList** control's **ListImages** or **ListImage** property.

```
Set img = iml.ListImages("open")
Set img = iml.ListImage("open")
```

See Also

ImageList

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Tabs Collection, Tab Object

Purpose

A **Tab** object represents an individual tab in the **Tabs** collection of a **TabStrip** control.

A **Tabs** collection contains a collection of **Tab** objects.

Syntax

tabstrip.**Tabs**(*index*)

tabstrip.**Tabs**.**Item**(*index*)

index:Variant. A value that identifies a Tab object in the Tabs collection. This may either be the Index property or the Key property of the desired Tab object.

Description

The **Tabs** collection can be accessed by using the standard collection methods, such as the **Item** method.

At run time, use the **TabStrip** control to insert and remove tabs, and use **Tab** object to specify any of these properties for a **Tab** object: **Caption**, **Image**, **ToolTipText**, **Tag**, **Index**, and/or **Key**.

Use the **Caption** and **Image** properties, separately or together, to label or put an icon on a tab.

To use the **Image** property, put an **ImageList** control on the form and fill the **ListImages** collection with **ListImage** objects, each of which has an index number and an optional key, if you add one. Set the **ImageList** property of the **TabStrip** control to associate it with the **TabStrip** control.

Use the **ToolTipText** property to temporarily display a string of text in a small rectangular box at run time when the user's cursor hovers over the tab.

To return a reference to a **Tab** object a user has selected, use the **SelectedItem** or **SelectedIndex** properties; to determine whether a specific tab is selected, use the **Selected** property. These properties are useful in conjunction with the BeforeClick event to verify or record data associated with the currently-selected tab before displaying the next tab the user selects.

Each **Tab** object also has read-only properties you can use to reference a single **Tab** object in the **Tabs** collection: **Left**, **Top**, **Height** and **Width**.

The **Tabs** collection properties and methods:

Add | Clear | Count | Item | Remove

The **Tab** properties and methods:

Caption | Height | Index | Image | Key | Left | hWnd | Ocx | Selected | TabStripName | Tag | Text | ToolTipText | Top | Width

Tab only properties:

OcxReturns an **Object** reference to the object that is attached to the **Tab**.

Example

```
Form Hidden Center frm1 = "TabStrip", , , 400, 300
```

```
Ocx TabStrip tbs = , 20, 20, ScaleWidth - 40,
  ScaleHeight - 40
Ocx Frame fr1 = "Tab #1"
Ocx Frame fr2 = "Tab #2"
Ocx Frame fr3 = "Tab #3"
Ocx Frame fr4 = "Tab #4"
OcxOcx fr1 Option opt1 = "Option #1", 20, 20, 80,
  24
OcxOcx fr1 Option opt2 = "Option #2", 20, 50, 80,
  24
OcxOcx fr2 CheckBox chk1 = "Check #1", 20, 20, 80,
  24
OcxOcx fr2 CheckBox chk2 = "Check #2", 20, 50, 80,
  24
OcxOcx fr3 TextBox txt1 = "TextBox #1", 20, 20,
  280, 40
OcxOcx fr3 TextBox txt2 = "TextBox #2", 20, 130,
  280, 40
OcxOcx fr4 Command cmd1 = "Command #1", 90, 20,
  80, 24
OcxOcx fr4 Command cmd2 = "Command #2", 90, 50,
  80, 24
tbs.Tabs.Add 1, , fr1.Caption , , fr1
tbs.AddItem 2, , fr2.Caption, , fr2
tbs.Add 3, , fr3.Caption, , fr3
tbs.AddItem 4, , fr4.Caption , , fr4
frm1.Show
tbs(2).Selected = True
Do
  Sleep
Until Me Is Nothing

Sub tbs_Change
  Switch tbs.SelectedIndex
  Case 1 : opt1.SetFocus
  Case 2 : chk1.SetFocus
  Case 3 : txt1.SetFocus
  Case 4 : cmd1.SetFocus
  EndSwitch
  Trace tbs.SelectedItem.Ocx
End Sub
```

Remarks

GFA-BASIC 32 specific

Instead of explicitly using the **Tabs** collection to access a **Tab** element, you can use a shorter notation. First, the **TabStrip** Ocx supports an **Item** property:

tbs.**Item**(idx)tbs.**Tabs**.**Item**(idx)

Like the **Item** method of tbs.**Tabs, Item** is the default method of **TabStrip**. Therefore, a **Tab** object can be accessed as follows:

tbs(idx)tbs.**Tabs**(idx)

tbs!idxtbs.**Tabs**!idx

Each dot saves about 30 bytes of code.

To enumerate over the **Tabs** collection of a **TabStrip** Ocx, use **For Each** on the Ocx control directly, like:

Local tab As Tab

For Each tab In tbs : DoSomething(tab) : Next

See Also

TabStrip

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

Menus

Purpose

To create, edit and delete window specific bar menus.

Syntax

Menu m$()

Menu idx, flags, txt$

state = Form.**MenuEnabled**
Form.**MenuEnabled** = state

retval = Form**.MenuItem**(idx)
Form**.MenuItem**(idx) = setval

Menu Kill

txt$ = Form**.MenuText**(idx)
Form**.MenuText**(idx) = txt$

Sub *Form*_**MenuEvent**(*[index%,] idx%*)

Sub *Form*_**MenuOver**(*[index%,] idx%*)

m$()	*: the string array containing the menu entries*
flags, idx, index	*: integer*
retval, setval	*: boolean, integer or string*
state	*: boolean*
txt$	*: string*

Description

Menu bars can be created in a window by using **Menu** m$() and, subsequently, edited using the **Menu** *idx, flags, txt$* command or the **MenuItem** and **MenuText** properties of the window itself. The enabled status of the menu itself (rather than the individual items) can be controlled using the **MenuEnabled** property, all menu events are handled by **MenuEvent** and **MenuOver** and the menu in the current window can be destroyed by the command **Menu Kill**.

Creating Menus using Menu *m$()* Show

Adding Items and Sub-Menus to Existing Menus Show

Editing Menu Item Properties using Menu *idx, flags, txt$* Show

Viewing and Setting Menu Item Properties using MenuItem(*idx*) Show

Viewing and Setting Menu Item Properties using APIs Show

Viewing and Setting Menu Labels using MenuText(*idx*) Show

Handling Menu Events Show

Removing Items from Menus Show

Disabling, Enabling and Destroying Menus Show

Examples

The example below creates a basic Menu and shows how items can be changed and events handled using standard GB32 commands. Show

This second example from the original German GFA help file, is a good illustration of how to mix standard GB32 commands and Windows APIs to create and alter menus. Show

Known Issues

As noted above, the **MenuItem** property of the Form hosting the menu does not work with menu items added through Windows APIs and this is because the **MenuItem** collection is an internal collection formed by GFA Basic which is created at the same time as the Menu and there is no programmatical means available to add to this collection once it has been created. See this article on Sjouke Hamstra's blog for a more detailed and technical explanation of this issue.

See Also

Popup

{Created by Sjouke Hamstra; Last updated: 20/12/2015 by James Gaite}

Forms Property (App)

Purpose

Returns a **Forms** collection, which is a collection whose elements represent each loaded form in an application.

Syntax

App.**Forms**

Description

The collection includes the application's MDI form, MDI child forms, and non-MDI forms. The **Forms** collection has a one method, **Item**, and a single property, **Count**, that specifies the number of elements in the collection.

Item(Index As Variant) is the default and returns a Form object.

Example

```
Debug.Show
OpenW 1
OpenW 33
Trace App.Forms.Count
Trace App.Forms(1).Name
Trace App.Forms.Item(2).Name
Trace App.Forms.Item(2).Caption
CloseW 1
CloseW 33
```

Remarks

You can also use **For Each** to enumerate over all forms.

```
Dim f As Form, n As Int
For n = 1 To 15 : OpenW Hidden n : Me.Caption =
  "Window " & n : Next n
Debug.Show
For Each f In App.Forms
  Trace f.Name
  Trace f.Caption
Next
For n = 1 To 15 : CloseW n : Next n
```

See Also

App, Controls

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Controls Property, Control Ocx

Purpose

Returns a reference to a collection of **Control** objects on a **Form**.

Syntax

Form.**Controls**

Control

Description

A collection of type **Control.** The collection can be iterated over using **For Each**. Furthermore, it provides the **Count** property.

The **Control** type as a generic variable type for controls. When you declare a variable **As Control**, you can assign it a reference to any control. You cannot create an instance of the **Control** class.

Example

```
Form frm1 = , 0, 0, 150, 200
// Populate Form
Ocx Command cmd = "Command", 10, 10, 100, 22
Ocx Option opt(1) = "Option 1", 10, 40, 100, 14
Ocx Option opt(2) = "Option 2", 10, 60, 100, 14
Ocx CheckBox checkbox = "Checkbox", 10, 85, 100,
 14
// Display Control properties in Debug screen
Debug.Show
~SetWindowPos(Debug.hWnd, 0, 250, 0, 500, 500, 0)
Dim o As Control
Trace frm1.Controls.Count
Debug.Print
For Each o In frm1.Controls
  Trace o.Name
  Try
    Trace o.Index
  Catch
  EndCatch
  Debug
Next
Do : Sleep : Until frm1 Is Nothing
Debug.Hide
```

Remarks

Accessing properties and methods of a control is faster if you use a variable declared with the same type as the control (for example, **As TreeView** or **As Command**), because GFA-BASIC 32 can use early binding. GFA-BASIC 32 must use late binding to access properties and methods of a control assigned to a variable declared **As Control**.

See Also

Form, Forms

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

Picture, StdPicture Object, Picture Property

Purpose

The **Picture** object enables you to manipulate bitmaps, icons, metafiles enhanced metafiles, GIF, and JPEG images assigned to objects having a **Picture** property. The **Picture** property returns or sets a graphic to be displayed in an Ocx object.

Syntax

Picture

StdPicture

*object***.Picture** [= *picture*]

object:Ocx object
picture:Picture object

Description

You frequently identify a **Picture** object using the **Picture** property of an object that displays graphics (such as a **Form** or **Image** control). If you have a **Form** control named frm1, you can set one **Picture** object equal to another using the **Set** statement, as shown in the example.

There are several ways to load a picture object:

- Use **LoadPicture** to load from disk. Specifically, you can load a bitmap from a BMP or DIB file. You can load an icon from an ICO file or load a metafile from a WMF file. You can load a cursor from an ICO file or a CUR file.

- Use **LoadPicture** with no argument to clear a picture file.

- Use **CreatePicture** with a GDI object handle from a Windows API function.

- Assign one **Picture** property from another **Picture** property, from an **Image** property, or from any other property with **StdPicture** or **Picture** type.

Dim X **As New StdPicture**

The **StdPicture** and **Picture** objects behave identical. **StdPicture** is a (co)class, while **Picture** is an interface. You can never create from an interface directly.

Properties

Name	**Type**	**Meaning**
Handle	Handle	Returns the handle to the graphic (bitmap, metafile, icon handle).
hPal	Handle	Returns the palette handle if available.
Height	Single	Returns the height of the picture in twips.
Width	Single	Returns the width of the picture in twips.
Type	Short	Returns the type of the picture (0=none, 1=bitmap, 2=metafile, 3=icon or cursor, 4=enhanced metafile).

Methods

Render *hDC As Handle, x, y, cx, cy As Long, xSrc, ySrc, cxSrc, cySrc As Single, lprcBounds As Handle*

hdc	The handle to the destination object's device context.
x, y	The x- and y-coordinate of upper left corner of the drawing region in the destination object. This coordinate is in the scale units of the destination object.
cx, cy	The width and height of drawing region in the destination object, expressed in the scale units of the destination object.
xSrc, ySrc	The x- and y-coordinate of upper left corner of the drawing region in the source object. This coordinate is in HIMETRIC units.
cxSrc, cySrc	The width and height of drawing region in the source object, expressed in HIMETRIC units.
lprcbounds	The world bounds of a metafile. This argument should be passed a value of Null unless drawing to a metafile, in which case the argument is passed a user-defined type corresponding to a RECT structure.

The recommended way to paint part of a graphic into a destination is through the **PaintPicture** method.

Example

```
Dim X As Picture
OpenW 1, 0, 0, 300, 300
OpenW 2, 300, 0, 300, 300 : AutoRedraw = 1
Color 255 : PCircle 100, 100, 50
Ocx Command cmd = "Transfer circle to Window 1",
 10, 200, 160, 22
```

```
Do : Sleep : Until Win_1 Is Nothing Or Win_2 Is
 Nothing
CloseW 1 : CloseW 2

Sub cmd_Click
  cmd.Visible = False
  Set X = Win_2.PrintPicture
  Win_1.Picture = X
EndSub
```

Remarks

When setting the **Picture** property at design time, the graphic is saved and loaded with the form. If you create an executable file, the file contains the image. When you load a graphic at run time, the graphic isn't saved with the application. Use the **SavePicture** statement to save a graphic from a form or picture box into a file.

See Also

LoadPicture, CreatePicture, PaintPicture, SavePicture

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

CreatePicture Function

Purpose

Creates a new picture object initialized with a GDI bitmap or icon handle.

Syntax

Set p = **CreatePicture**(hBmp, Owner)

p:Picture
hBmp:Handle
Owner:Bool

Description

The *hBmp* parameter specifies the GDI handle for a bitmap (BMP or DIB) or a icon.

The *Owner* parameter indicates whether the picture is to own the GDI picture handle for the picture it contains, so that the picture object will destroy its picture when the object itself is destroyed. When Owner = 1 the Picture object takes ownership.

Example

Example - Icon

```
Dim p As Picture
Dim h% = LoadIcon(_INSTANCE, 1)
Set p = CreatePicture(h, 1)
PaintPicture p, 1, 1
```

Example - Bitmap

```
OpenW 1
Local i%, h%, p As Picture
For i = 0 To 255 Step 2
  Color RGB(i, i * 2, i * 3)
  Circle 100, 100, i / 2
Next i
Get 0, 0, 200, 200, h// get the bitmap
Color 0
Set p = CreatePicture(h, 1)
Set Win_1.Picture = p
Win_1.PictureMode = 1
Win_1.Refresh
```

Remarks

See Also

Picture, PaintPicture, SavePicture

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

LoadPicture Function

Purpose

Loads a graphic and returns a **Picture** object

Syntax

LoadPicture([*filename$*] [, *x*, *y*, *c*]**)**

filename$:sexp
x, y, c:iexp

Description

LoadPicture loads a picture file indicated by *filename*$ and returns a Picture object. The return value can be assigned directly to object properties that take a .Picture reference, using **Set** =. Examples are Form.Picture, *Ocx*.MouseIcon, Command.Picture, etc.

Graphics formats recognized by **LoadPicture** include bitmap (.bmp) files, icon (.ico) files, run-length encoded (.rle) files, metafile (.wmf) files, enhanced metafiles (.emf), GIF (.gif) files, and JPEG (.jpg) files.

Graphics are cleared from forms, picture boxes, and image controls by assigning **LoadPicture** with no argument.

To assign an icon to a form, set the return value of the **LoadPicture** function to the **Icon** property of the **Form** object: Set Form.Icon = **LoadPicture**(icon$).

The optional arguments *x, y,* and *c* are used only with icon and cursor files. These icon files often contain several images with different size and color formats. To get the required icon from several different ones, the **LoadPicture** takes size and color arguments:

Set Form.SmallIcon = **LoadPicture**("ico-name.ico", x, y, c)

If *filename* is a cursor or icon file, and either *x* or *y* is specified, the *x* and *y* specify the width or height desired. In a file containing multiple separate images, the best possible match is used if an image of that size is not available. X and y values are only used when c (*color depth*) is > 1. For icon files 255 is the maximum possible value.

If both *x* =0 and *y* = 0, a small icon will be loaded (mostly 16x16).
If *x* or *y* is equal 0 and *y* or *x* = 1, the default 32x32 large icon will be loaded (actually determined by the video driver).

If *c* = 0 the default colors are used and a best available match is made.
If *c* = 1 a monochrome image is searched and loaded.
If *c* => 2 searches for an image with the specified number of colors.

Example

```
Local bmp As Picture, n As Int32
Ocx Form test
AutoRedraw = True
// Find picture file
Local d$ =
  GetSetting("\\HKEY_CLASSES_ROOT\Applications\GfaW
  in32.exe\shell\open\command", , "")
If Left(d$, 1) = #34 Then d$ = Mid(d$, 2)
```

```
n = RInStr(d$, "\") : If n <> 0 Then d$ = Left(d$,
  n - 1)
d$ = d$ & "\..\samples\bitmaps\splash.bmp"
Print d$
If Not Exist(d$) Then _
  MsgBox("Can not locate Splash.bmp
    file"#13#10#13#10"Please manually place it in
    the GFABASIC32\Samples\Bitmaps folder and try
    again.")  : End
// Load the picture
Set bmp = LoadPicture(d$)
// Show the picture and stretch it over the whole
  form
PaintPicture bmp, 0, 0, _X, _Y
Do
  Sleep
Until Me Is Nothing      // Alt + F4
Set bmp = Nothing
```

Remarks

Pictures stored in the :File section can be loaded with **LoadPicture** but it should be noted that, to achieve this, GFABasic copies it to the temporary directory first and then loads it into memory from this newly created physical file. This is due to the fact that **LoadPicture** can not handle the 'unpacking' of the file. When loading big files (BMP, JPEG, GIF) this will increase the load time, although with technology as it is, this may not be either significant or noticeable.

GFA-BASIC 32 also supports the conversion of normal API bitmaps to an OLE Picture object with the **CreatePicture** function.

See Also

PaintPicture, SavePicture, CreatePicture

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

PaintPicture Method

Purpose

Draws the contents of a graphics file (.bmp, .wmf, .emf, .cur, .ico, or .dib) on a **Form** or **Printer**.

Syntax

[object.]**PaintPicture** pict, x1, y1, w1, h1, x2, y2, w2, h2, opcode

x1, y1, w1, h1, x2, y2, w2, h2:floating-point exp
opcode:iexp

Description

object. - The name of the **Form** or **Printer** object where the picture is to be placed. This argument is optional. If it's omitted, the form with the focus (Me) is assumed.

pict - The **Picture** object to paint.

x1, y1 - Single-precision values indicating the destination coordinates (in other words, the location on the destination object where the top-left corner of the image is to be drawn). The ScaleMode property of the object determines the unit of measure used.

w1, h1 - Single-precision values indicating the destination width and height of the picture, using units specified by the ScaleMode property of the destination object. If the destination width and/or height is larger or smaller than the source width (w2) or height (h2), the picture is stretched or compressed to fit. These arguments are optional; if you omit them, the source width (w1) and height (h1) are used with no stretching or compression.

x2, y2 - Single-precision values indicating the source coordinates of the region in the source object that is to be copied (in units specified by the source object's ScaleMode property). These arguments are optional; if you omit them, 0 is assumed (indicating the top-left corner of the source image).

w2, h2 - Single-precision values indicating the width and height of the region within the source that is to be copied (in units specified by the source object's ScaleMode property). These arguments are optional; if you omit them, the entire source width and height are used.

opcode - A type Long value that defines the bit-wise operation that is performed between the pixels of the source picture and the pixels of any existing image on the destination. This argument, which is optional, is useful only with bitmaps. If you omit the argument, the source is copied onto the destination, replacing anything that is there.

For a complete list of bit-wise operator constants, see the **BitBlt** RasterOp Constants

PaintPicture without an object identifier is executed on the current output device (**Me** or **OutPut** =)

Example

```
OpenW # 1 : AutoRedraw = 1
Local pic As Picture, h As Handle, n As Int32
For n = 1 To 601 Step 50
  Line 0, n, 601, n
  Line n, 0, n, 601
Next n
Set pic = Win_1.PrintPicture
Dlg Print Win_1, 0, h
If h <> 0
  Local Int32 ht = HimetsToPixelY(pic.Height), wd =
    HimetsToPixelX(pic.Width)
  SetPrinterHDC h
  Output = Printer
  'Lprint ""; // Causes an error with some printers
    if used to force start the print process
  Printer.StartDoc "test"
  Printer.StartPage
  PaintPicture pic, 0, 0, wd * 2, ht * 2
  Printer.EndPage
  Printer.EndDoc
  Output = Me
EndIf
```

This prints a hardcopy of a form (Me) as small as a stamp on the printer, but you can scale it by changing the *wd*2* and *ht*2* parameters as you wish.

Remarks

See Also

Bitblt

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

SavePicture Command

Purpose

Saves a **Picture** object to a file.

Syntax

SavePicture picture, file$

picture:Picture Object
file$:sexp, filename

Description

Saves a graphic from the **Picture** or **Image** property of an object or control (if one is associated with it) to a file.

If a graphic was loaded from a file to the Picture property of an object, either at design time or at run time, and it’s a bitmap, icon, metafile, or enhanced metafile, it's saved using the same format as the original file. If it is a GIF or JPEG file, it is saved as a bitmap file.

Graphics in an **Image** property are always saved as bitmap (.bmp) files regardless of their original format.

Interesting, is the possibility to save the **AutoRedraw** bitmap, because the Picture object is returned with the **Image** property.

Example

```
OpenW 1, 0, 0, 200, 200
```

```
AdjustW 1, 200, 200
AutoRedraw = 1
For i = 0 To 500
  Color RGB(i * 5, i * 6, i * 7)
  Circle 100, 100, i
Next
Global i%
SavePicture Me.Image, "c:\Test.Bmp"
Do
  Sleep
Until Me Is Nothing
Kill "c:\test.bmp"  // Tidy-up line
```

See Also

Picture, Form

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

GUID Command

Purpose

Declares or generates a GUID value literal (constant).

Syntax

GUID name = value

GUID name **= new**

Description

Like **Const** the **GUID** command declares a constant GUID value, where *name* specifies the name for the constant variable. A GUID constant points to an address, the actual value is located at *****name or **V:**name.

GFA-BASIC 32 can also generate a unique GUID when **new** is used, rather than a value. After leaving the code line, GFA-BASIC 32 adds a new GUID name in the place of the keyword **new**.

Example

```
// by typing in GUID test = new you get
  immediately:
'
GUID test = d6f0dbc0-11d3-bdd1-9f15-0000e85cfc38
'
Type GUID
  D1 As Int
  D2 As Card
  D3 As Card
  D4(7) As Byte
EndType
Local f As GUID
// convert to string for output
Print GUID$(V:f)
Print GUID$(V:test)
// Format of a 'clear text' GUID (128 bit)
// 8characters-4characters-4characters-
// 4characters-12characters
Do
  Sleep
Until Me Is Nothing
```

See Also

GUID$

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

GUID$ Function

Purpose

Converts a binary **GUID** into a string.

Syntax

$ = GUID$(addr%**)**

Description

With the function **GUID$**(addr) you convert a binary **GUID** into a string. The parameter *addr* is the address of a 16 byte value. The result will be a string without "{}", converted to lowercase (0-9a-f), in the usual GUID format.

Example

```
GUID test = d6f0dbc0-11d3-bdd1-9f15-0000e85cfc38
Print GUID$(V:test)
// result: d6f0dbc0-11d3-bdd1-9f15-0000e85cfc38
// Format of a 'clear text' GUID (128 bit)
// 8characters-4characters-4characters-
// 4characters-12characters
```

See Also

GUID

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

vbDeleteSetting Command

Purpose

Deletes a section or a key setting from an entry in the Windows registry using VB compatible registry commands.

Syntax

vbDeleteSetting *appName$, [section$] [, key$]*

Description

Deletes a section or key setting in the Visual Basic standard registry location for storing program information for applications created in Visual Basic:

HKEY_CURRENT_USER\Software\VB and VBA Program Settings\appName\section\key

The registry stores data in a hierarchically structured tree. Each node in the tree is called a *key*. Each key can contain both *subkeys* and data entries called *values*.

The **vbDeleteSetting** function syntax has these arguments:

appName - Required. String expression containing the name of the application or project to which the section or key setting applies. May include *section* in GFA-BASIC 32.

section - Optional. String expression containing the name of the section where the key setting is found. If only *appName* and *section* are provided, the specified section is deleted along with all related key settings.

key - Optional. String expression containing the name of the key setting to return.

Example

```
vbSaveSetting "MyApp", "Startup", "Top", 75
vbSaveSetting "MyApp", "Startup", "Left", 50
Debug vbGetSetting("MyApp", "Startup", "Left", ,
  25)
vbDeleteSetting "MyApp", "Startup"
vbDeleteSetting "MyApp"
```

Remarks

vbDeleteSetting does not work in the same fashion as they do with non-nested keys. This means that the command won't delete subkeys recursively. Use more than one **vbDeleteSetting** statement to remove sections of the nested keys before removing the top level key, rather than attempting to remove the top key in isolation. See example.

See Also

vbDeleteSetting, vbGetSetting, vbGetSettingType, GetSetting, GetSettingType, SaveSetting, DeleteSetting, CreateRegKey, OpenRegKey, CloseRegKey, GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount, GetRegSubKey, GetRegSubKeyCount

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

vbGetSetting, vbGetSettingType Function

Purpose

Returns a key setting value or type from an entry in the Windows registry using VB compatible registry commands.

Syntax

$ = **vbGetSetting**(*appName$, [section$], key$ [, ,default$]*)

% = **vbGetSettingType**(*appName$, [section$], [key$]*)

Description

These functions are VB compatible and return registry entries. The registry stores data in a hierarchically structured tree. Each node in the tree is called a *key*. Each key can contain both *subkeys* and data entries called *values*.

Visual Basic applications are required to store their registry settings under the entry called *HKEY_CURRENT_USER\Software\VB* and *VBA Program Settings\appName\ section.*

The **vbGetSetting**, **vbGetSettingType**, **vbSaveSetting**, **vbDeleteSetting** functions take appName and section as parameters to access the required entry in the *HKEY_CURRENT_USER\Software\VB* and *VBA Program Settings*\ registry key.

The **vbGetSetting** function syntax has these arguments:

appName - Required. String expression containing the name of the application or project whose key setting is requested. May include section in GFA-BASIC 32.

section - Optional. String expression containing the name of the section where the key setting is found.

key - Required. String expression containing the name of the key setting to return.

,default - Optional. Expression containing the value to return if no value is set in the key setting. If omitted, default is assumed to be a zero-length string ("").

If any of the items named in the **vbGetSetting** arguments do not exist, **vbGetSetting** returns the value of default.

The **vbGetSettingType** can be used to obtain the data type of the value of *key$*. Normally, the counter part of **vbGetSetting**, the VB compatible command **vbSaveSetting** saves data always in the string (REG_SZ) format.

Example

```
vbSaveSetting "MyApp", "Startup", "Top", 75
vbSaveSetting "MyApp", "Startup", "Left", 50
Debug vbGetSetting("MyApp", "Startup", "Left", ,
 25)
Debug vbGetSettingType("MyApp", "Startup", "Left")
vbDeleteSetting "MyApp", "Startup"
vbDeleteSetting "MyApp"
```

Remarks

It is possible to use the **vbGetSetting** statement to retrieve values form nested levels of keys and values in the Registry. This behavior is desirable in some cases.

For example, when receiving the location of a SYSTEM.MDA file, the Access engine expects the SystemDB value to exist in a subkey of Engines\Jet, like this:

HKEY_CURRENT_USER
\Software
\VB and VBA Program Settings
\MyApp
\Engines
\Jet
SystemDB = c:\access\system.mda

You can obtain nested levels in the Registry by using this syntax:

```
f$ = vbGetSetting("MyApp", "Engines\Jet",
 "SystemDB")
```

vbDeleteSetting does not work in the same fashion as they do with non-nested keys.

See Also

vbSaveSetting, vbDeleteSetting, GetSetting, GetSettingType, SaveSetting, DeleteSetting, CreateRegKey, OpenRegKey, CloseRegKey, GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount, GetRegSubKey, GetRegSubKeyCount

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

vbSaveSetting Command

Purpose

Saves or creates an application entry in the Windows registry entry using a VB compatible registry command.

Syntax

vbSaveSetting *appName$, [section$], key$, value*

Description

This VB compatible command saves a value under the *key* setting in the registry. The registry stores data in a hierarchically structured tree. Each node in the tree is called a *key*. Each key can contain both *subkeys* and data entries called *values*.

Visual Basic applications are required to store their registry settings under the entry called *HKEY_CURRENT_USER\Software\VB* and *VBA Program Settings\appName\ section.*

The **vbGetSetting**, **vbGetSettingType**, **vbSaveSetting**, **vbDeleteSetting** functions take *appName* and *section* as parameters to access the required entry in the *HKEY_CURRENT_USER\Software\VB* and *VBA Program Settings* registry key.

The **vbSaveSetting** command syntax has these arguments:

appName - Required. String expression containing the name of the application or project whose key setting is requested. May include *section* in GFA-BASIC 32.

section - Optional. String expression containing the name of the section where the key setting is found.

key - Required. String expression containing the name of the key setting to return.

value - Expression containing the value that *key* is being set to.

vbSaveSetting saves data always in the string (REG_SZ) format.

Example

```
vbSaveSetting "MyApp", "Startup", "Top", 75
vbSaveSetting "MyApp", "Startup", "Left", 50
Debug vbGetSetting("MyApp", "Startup", "Left", ,
 25)
vbDeleteSetting "MyApp", "Startup"
vbDeleteSetting "MyApp"
```

Remarks

It is possible to use the **vbSaveSetting** statement to create nested levels of keys and values in the Registry. This behavior is desirable in some cases.

For example, when receiving the location of a SYSTEM.MDA file, the Access engine expects the SystemDB value to exist in a subkey of Engines\Jet, like this:

HKEY_CURRENT_USER
\Software

\VB and VBA Program Settings
\MyApp
\Engines
\Jet
SystemDB = c:\access\system.mda

You can create nested levels in the Registry by using this syntax:

```
SaveSetting "TestApp", "Test2\Test3", "TestVal",
  "TestSetting"
```

This will create a section of the Registry that looks like:

HKEY_CURRENT_USER
\Software
\VB and VBA Program Settings
\TestApp
\Test2
\Test3
TestVal = TestSetting

To retrieve values stored in the Registry like this, use the same syntax with the **vbGetSetting** function. Some restrictions are inherited when creating nested keys with **vbSaveSetting**.

vbDeleteSetting does not work in the same fashion as they do with non-nested keys.

See Also

vbDeleteSetting, vbGetSetting, vbGetSettingType, vbDeleteSetting, GetSetting, GetSettingType, SaveSetting, DeleteSetting, CreateRegKey, OpenRegKey, CloseRegKey, GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount, GetRegSubKey, GetRegSubKeyCount

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

DeleteSetting Command

Purpose

Deletes a subkey or a value in the Registry.

Syntax

DeleteSetting *hkey$, [subkey$] [,value$]*

Description

Deletes a key, subkey or value.

The registry stores data in a hierarchically structured tree. Each node in the tree is called a *key*. Each key can contain both *subkeys* and data entries called *values*.

The **DeleteSetting** command syntax has these arguments:

hkey$	Required. String expression containing the name of the application or project to which the section or key setting applies. The value is saved under "\\hkcu\Software\" + *hkey$*. May include *subkey$*.
subkey$	Optional. String expression containing the name of the section where the key setting is stored. By default the value is saved under "\\hkcu\Software\" + *hkey$* + "\" + *subkey$. hkey$* may include *subkey$*; *subkey$* is then omitted.
value$	Optional. String expression containing the name of the key setting to delete.

When *value$* = "" or when the *value$* parameter is omitted the subkey will be deleted, however it must not have subkeys. GFA-BASIC 32 checks for the existence of descendants before deleting a subkey and a run time error occurs if there are.

Example

```
SaveSetting "MyApp", "Startup\New", "Top", 75
SaveSetting "MyApp", "Startup\New", "Left", 50
DeleteSetting "MyApp", "Startup\New"
DeleteSetting "MyApp", "Startup"
DeleteSetting "MyApp"
```

This example first creates nested levels in \\HKCU\Software\MyApp and then deletes the nested levels one by one.

Remarks

DeleteSetting conforms to the API function *RegDeleteKey().*

Windows NT and later have a built-in protection against deleting a subkey containing subkeys, Windows 95, 98, Me don't. The protection in **DeleteSetting** is necessary because recursive deletion of keys may cause disaster in case of an error in the parameters.

See Also

vbDeleteSetting, vbGetSetting, vbGetSettingType, vbDeleteSetting, GetSetting, GetSettingType, SaveSetting, DeleteSetting, CreateRegKey, OpenRegKey, CloseRegKey, GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount, GetRegSubKey, GetRegSubKeyCount

{Created by Sjouke Hamstra; Last updated: 30/09/2014 by James Gaite}

GetSetting, GetSettingType Function

Purpose

Returns a key setting value or type from an entry in the Windows registry.

Syntax

$ = **GetSetting**(*hkey$, [subkey$], [name$] [, **Str** | Int | Bin] [,default]*)

% = **GetSettingType**(*hkey$, [subkey$], [name$]*)

Description

The registry stores data in a hierarchically structured tree. Each node in the tree is called a *key*. Each key can contain both *subkeys* and data entries called *values*.

The **GetSetting** function syntax has these arguments:

hkey$	Required. String expression containing the name of the application or project to which the section or key setting applies. The value is saved under "\\hkcu\Software\" + *hkey$*. May include *subkey$*.
subkey$	Optional. String expression containing the name of the section where the key setting is stored. By default the value is saved under "\\hkcu\Software\" + *hkey$* + "\" + *subkey$.*

	hkey$* may include *subkey$; *subkey$* is then omitted.
name$	Optional. String expression containing the name of the key setting to return. If omitted the default value (Standard) is returned.
type	Optional. Besides the default type **Str,** the **Int** and **Bin** data types are allowed. In case of **Bin**, the data is returned in a string. To restore the data to a user-defined type copy the string data to the udt using: **Poke$ V:** udt, value$
default	Optional. Expression containing the value to return if no value is set in the key setting.

When an integer value is read as a **Str**, the integer is converted using **Dec$**, internally. Reading an integer as **Bin** will convert the 4 bytes to a string using **Mkl$**(). When a **Str** is read as an **Int**, **Val**() is applied. When a **Bin** value is read as an integer, only the first 4 bytes ar read as numeric value. **Str** and **Bin** are equivalent.

In case of an error, the default value is returned. If omitted, default is assumed to be a zero-length string ("") or 0.

GetSetting can also be used to read other keys than \\hkcu\Software only.

```
Print GetSetting("\\hkcr\.g32", , "")
```

Returns value for the "Standard" entry: "G32File"

When *hkey$* starts with "\\" a predefined reserved handle must follow:

"\\HKEY_CLASSES_ROOT" or "\\hkcr" or "\\80000000"
"\\HKEY_CURRENT_CONFIG"

"\\HKEY_CURRENT_USER" or "\\hkcu"
"\\HKEY_LOCAL_MACHINE" or "\\hklm"
"\\HKEY_USERS"
"\\HKEY_PERFORMANCE_DATA" (Windows NT)
"\\HKEY_DYN_DATA" (Windows 95 and Windows 98)

The *hkey$* parameter can be assembled using "\\" & Hex(HKEY_CLASSES_ROOT)

```
Print GetSetting("\\" & Hex(HKEY_CLASSES_ROOT) &
  "\.g32", , "")
```

When *hkey$* starts with "\" it must be followed with a valid key for HKEY_CURRENT_USER, because "\" determines a descendant of hkcu. For instance, the following statements return the same value

```
a$ = GetSetting("\Software\Firma\prog", , "name")
a$ = GetSetting("Firma", "prog", "name")
```

The *hkey$* parameter may also specify the handle to a registry key obtained using **OpenRegKey**, see example.

GetSettingType returns the data type for the registry value. The returns value is 1 (REG_SZ) for a string, 3 (REG_BINARY) for binary data, 4 (REG_DWORD) for an **Int**-value, or 0 (REG_NONE) in case of an error.

```
Print GetSettingType("\\hkcr\.g32", , "") //
  returns the Standard name data type: 1 (REG_SZ)
```

Example

```
PrintWrap = 1
Local hkey$, value$, i%, t#
// this selects all values in the key and returns
  first the time
```

```
// for the access with OpenRegKey, after without
Local key$ = "\\HKEY_LOCAL_MACHINE\Software" _
  "\Microsoft\Windows\CurrentVersion"
If IsWinNT     // GetVersion() > 0
  key$ = "\\HKEY_LOCAL_MACHINE\Software" _
    "\Microsoft\Windows NT\CurrentVersion"
End If
Print "OpenRegKey + GetSetting"
t = Timer
Restore
hkey$ = OpenRegKey(key$)
For i% = 1 To 50
  Read value$
  Exit If value$ = "@"
  Write GetSetting(hkey$, , value$);
  Print ", ";
Next
~CloseRegKey(hkey$)
Print : Print Timer - t
Print "GetSetting only"
t = Timer
Restore
hkey$ = key$
For i% = 1 To 50
  Read value$
  Exit If value$ = "@"
  Write GetSetting(hkey$, , value$);
  Print ", ";
Next
Print : Print Timer - t
Print "OpenRegKey + GetRegValCount + GetRegVal"
t = Timer
hkey$ = OpenRegKey(key$)
For i% = 1 To GetRegValCount(hkey$)
  Write GetRegVal(hkey$, i);
  Print ", ";
Next
```

```
CloseRegKey hkey$
Print : Print Timer - t
// This is a list of the value names for the
  Registry
// directory on a Windows 98 computer.
Data InstallType,SetupFlags,DevicePath, _
  ProductType,RegisteredOwner
Data RegisteredOrganization,ProductId, _
  LicensingInfo,DVD_Region,BPC_Region
Data OldWinVer,SubVersionNumber, _
  ProgramFilesDir,CommonFilesDir,WallPaperDir
Data MediaPath,ConfigPath,SystemRoot, _
  OldWinDir,ProductName,Registration _ ExtDLL
Data RegDone,FirstInstallDateTime,Version, _
  VersionNumber,PiFirstTime Only,ProductKey
Data DigitalProductId,AuditMode, _
  ProgramFilesPath,SM_AccessoriesName, _
  PF_AccessoriesName
Data HWID,OtherDevicePath,ChannelFolderName, _
  LinkFolderName,Plus! VersionNumber
Data BootCount,@
```

See Also

vbDeleteSetting, vbGetSetting, vbGetSettingType, vbDeleteSetting, GetSetting, GetSettingType, SaveSetting, DeleteSetting, CreateRegKey, OpenRegKey, CloseRegKey, GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount, GetRegSubKey, GetRegSubKeyCount

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

SaveSetting Command

Purpose

Saves or creates an application entry in the Windows registry.

Syntax

SaveSetting *hkey$, [subkey$], name$, [Int | Bin | Str,] value*

Description

Saves a value in the registry under \\HKEY_CURRENT_USER\Software or any other node.

The registry stores data in a hierarchically structured tree. Each node in the tree is called a *key*. Each key can contain both *subkeys* and data entries called *values*.

The **SaveSetting** command syntax has these arguments:

hkey$	Required. String expression containing the name of the application or project to which the section or key setting applies. The value is saved under "\\hkcu\Software\" + *hkey$*. May include *subkey$*.
subkey$	Optional. String expression containing the name of the section where the key setting is stored. By default the value is saved under "\\hkcu\Software\" + *hkey$* + "\" + *subkey$. hkey$* may include *subkey$*; subkey$ is then omitted.

name$	Required. String expression containing the name of the key setting to set. If set to "" the default value (Standard) is written.
type	Optional. Besides the default type **Str,** the **Int** and **Bin** data types are allowed. In case of **Bin**, the data must be stored in a string. To save a user-defined type copy the binary data to a string using: value$ = **Peek$**(**V:**udt, **SizeOf**(udt))
value	Expression containing the value that *key* is being set to.

SaveSetting can also be used to write to other keys than \\hkcu only. When *hkey$* starts with "\\" a predefined reserved handle must follow.

"\\HKEY_CLASSES_ROOT" or "\\hkcr" or "\\80000000" (see Known Issues)
"\\HKEY_CURRENT_CONFIG"
"\\HKEY_CURRENT_USER" or "\\hkcu"
"\\HKEY_LOCAL_MACHINE" or "\\hklm"
"\\HKEY_USERS"
"\\HKEY_PERFORMANCE_DATA" (Windows NT)
"\\HKEY_DYN_DATA" (Windows 95 and Windows 98)

The *key$* parameter can be assembled using "\\" & Hex(HKEY_CLASSES_ROOT)

When *hkey$* starts with "\" it must be followed with a valid key for HKEY_CURRENT_USER, because "\" determines a descendant of hkcu. For instance, **SaveSetting** "\Software\Company\prog", , "name", "123" writes the same value as **SaveSetting** "Company", "prog", "name", "123".

Example

1. Save, get, and delete application settings.

```
SaveSetting "MyApp", "Startup\New", "Top", 75
SaveSetting "MyApp", "Startup\New", "Left", 50
Debug.Print GetSetting("MyApp", "Startup\New",
  "Left")
MsgBox "Open Registry to verify settings."
DeleteSetting "MyApp", "Startup\New"
DeleteSetting "MyApp", "Startup"
DeleteSetting "MyApp"
```

2. Create and delete a file association

```
Dim ext$     = ".zzz"
Dim cmdkey$  = "MyGFA32App.Document"
Dim descr$   = "MyGFA32App Document"
Dim appPath$ = App.FileName & " %1"
SaveSetting
  "\\HKEY_CURRENT_USER\Software\Classes", ext$, ""
  ,  cmdkey$
SaveSetting
  "\\HKEY_CURRENT_USER\Software\Classes", cmdkey$,
  "", descr$
SaveSetting
  "\\HKEY_CURRENT_USER\Software\Classes\" +
  cmdkey$, "shell\open\command", "", appPath$
MsgBox "Open Registry to verify settings."
'Remove File Association
DeleteSetting "\\hkcr", ext$
DeleteSetting "\\hkcr\" + cmdkey$,
  "shell\open\command"
DeleteSetting "\\hkcr\" + cmdkey$, "shell\open"
DeleteSetting "\\hkcr\" + cmdkey$, "shell"
DeleteSetting "\\hkcr", cmdkey$
```

Remarks

Note - **SaveSetting** creates a key when it doesn't exist, even in HKEY_CLASSES_ROOT or "\\hkcr". It is not necessary to use the GFA-BASIC 32 **CreateRegKey** function to first create the key.

DeleteSetting conforms to the Api function *RegDeleteKey().* It removes subkeys only, i.e. you must specify the parent key if you want to remove a key. Always delete keys step by step, because Windows NT does not support removing of nested keys.

Known Issues

In later versions of Windows (certainly from Windows 8 onwards), you can get an error saving a setting directly to \\hkcr or \\HKEY_CURRENT_ROOT if the key does not exist. This is due to changes in Windows security protocols.

To get around this problem, instead of using \\hkcr, use the longer \\HKEY_CURRENT_USER\Software\Classes instead. The latter is actually the source of the former in any one user account and any changes made will be reflected in \\hkcr.

See Also

vbDeleteSetting, vbGetSetting, vbGetSettingType, vbSaveSetting, GetSetting, GetSettingType, DeleteSetting, CreateRegKey, OpenRegKey, CloseRegKey, GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount, GetRegSubKey, GetRegSubKeyCount

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

CreateRegKey, OpenRegKey, CloseRegKey Functions

Purpose

Create, open and close a registry entry.

Syntax

hkey$ = **CreateRegKey**(*key$[, subkey$]*)

hkey$ = **OpenRegKey**(*key$[, subkey$]*)

r% = **CloseRegKey**(*hkey$*)

CloseRegKey *hkey$*

Description

Before an application can add data to the registry, it must create or open a key. To create or open a key, an application always refers to the key as a subkey of a currently open key. A GFA-BASIC 32 application can use the **OpenRegKey** function to open a key and the **CreateRegKey** to create a key. The return value *hkey$* contains the handle to the opened or created *key\subkey* and is used as the first parameter in GFA-BASIC's other low-level registry functions.

To close a key and write the data it contains into the registry you can use the GFA-BASIC 32 function **CloseRegKey**, which is also available as a command. **CloseRegKey** takes the return value of **CreateRegKey** or **OpenRegKey**.

The *key$* must specify one of the following predefined reserved handle values (note three of them can be shortened):

"\\HKEY_CLASSES_ROOT" or "\\hkcr" or "\\80000000"
"\\HKEY_CURRENT_CONFIG"
"\\HKEY_CURRENT_USER" or "\\hkcu"
"\\HKEY_LOCAL_MACHINE" or "\\hklm"
"\\HKEY_USERS"
"\\HKEY_PERFORMANCE_DATA" (Windows NT)
"\\HKEY_DYN_DATA" (Windows 95 and Windows 98)

The *key$* parameter can be assembled using "\\" & Hex(HKEY_CLASSES_ROOT)

The *key$* may be concatenated with the *subkey$*.

The *subkey$* parameter is optional (because it may be combined with *key$*) specifies the name of a key that is to be opened or created. This key must be a subkey of the key identified by the *key$* parameter.

The return value is a key handle as a hexadecimal string preceded with \\, for instance "\\80000000".

Example

```
PrintWrap = 1
Local hkey$, i%
Print "Installed software:"
hkey$ = OpenRegKey("\\HKEY_CURRENT_USER\Software")
For i% = 1 To GetRegSubKeyCount(hkey$)
  Write GetRegSubKey(hkey$, i);
  Print ", ";
Next
CloseRegKey hkey$
```

Remarks

The GFA-BASIC 32 functions **CreateRegKey, OpenRegKey**, and **CloseRegKey** conform to the API functions *RegCreateKey(),RegOpenKey(), and RegCloseKey(), respectively.*

See Also

vbSaveSetting, vbDeleteSetting, vbGetSettingType, vbGetSetting, GetSetting, GetSettingType, SaveSetting, DeleteSetting, CreateRegKey, OpenRegKey, CloseRegKey, GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount, GetRegSubKey, GetRegSubKeyCount

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount Functions

Purpose

These functions enumerates the values for the specified open registry key.

Syntax

$ = **GetRegVal**(*hkey$, idx% [, Int | Bin | **Str**]*)

$ = **GetRegValName**(*hkey$, idx%*)

% = **GetRegValType**(*hkey$, idx%*)

% = **GetRegValNameCount**(*hkey$*)

Description

The **GetRegVal-** functions are used to enumerate the values for the specified open registry key.

The first parameter *hkey*$ of all functions specifies a key handle obtained with the GFA-BASIC 32 function **OpenRegKey**.

The *idx%* parameter specifies the index of the value to retrieve. This parameter should be one (1) for the first call to any of the **GetRegVal** functions and then be incremented for subsequent calls, until **GetRegValNameCount** is reached.

GetRegValName obtains the name of the value with index *idx%*. **GetRegValType** obtains the type code for the value entry with index *idx%*. **GetRegVal** obtains the data for the value entry with index *idx%*.

Example

```
DisplayCurrWinVerReg

Sub DisplayCurrWinVerReg
  Local x$, i%, j%, hkey$
  Local key$ = "\\HKEY_LOCAL_MACHINE\Software" _
    "\Microsoft\Windows\CurrentVersion"
  If IsWinNT      // GetVersion() > 0
    key$ = "\\HKEY_LOCAL_MACHINE\Software" _
      "\Microsoft\Windows NT\CurrentVersion"
  End If
  hkey$ = OpenRegKey(key$)
  For i% = 1 To GetRegValCount(hkey$)
    Print i; Tab(5);
    Print GetRegValName(hkey$, i); Tab(30);
    Switch GetRegValType(hkey$, i)
    Case REG_SZ
      Print "Str"; Tab(36);
      Write GetRegVal(hkey$, i)
    Case REG_DWORD
      Print "Int"; Tab(36);
      Write GetRegVal(hkey$, i, Int)
    Case REG_BINARY
      x$ = GetRegVal(hkey$, i, Bin)
      Print "Bin"; Tab(36);
      For j = 1 To Min(Len(x$), 32)
        Print Hex(Asc(x$, j), 2); " ";
      Next
      Print
    Default
      Print "Type="; GetRegValType(hkey$, i);
        Tab(36); _
        GetRegVal(hkey$, i)
    EndSwitch
  Next
  CloseRegKey hkey$
EndSub
```

Remarks

The **GetRegVal, GetRegValName, GetRegValType** conform to the API function *RegEnumValue()*. Because this API function is called separately for each function, a little overhead is created, however this is only minimal.

GetRegValNameCount(*hkey$*) conforms to the API function *RegQueryInfoKey(,,,lpcSubKeys,..),* which retrieves information about a specified registry key.

See Also

vbSaveSetting, vbDeleteSetting, vbGetSettingType, vbGetSetting, GetSetting, GetSettingType, SaveSetting, DeleteSetting, CreateRegKey, OpenRegKey, CloseRegKey, GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount, GetRegSubKey, GetRegSubKeyCount

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

GetRegSubKey, GetRegSubKeyCount Functions

Purpose

To enumerate subkeys.

Syntax

$ = **GetRegSubKey**(*hkey$, idx%*)

% = **GetRegSubKeyCount**(*hkey$*)

Description

To enumerate subkeys, an application should initially call the **GetRegSubKeyCount** to obtain the number of subkeys for a specified *hkey$*. The application should call **GetRegSubKey** setting idx% to 1 and then increment the *idx%* parameter and call **GetRegSubKey** until the number of subkeys is reached.

Example

```
PrintWrap = 1
Local hkey$, i%
Print "Installed software:"
hkey$ = OpenRegKey("\\HKEY_CURRENT_USER\Software")
For i% = 1 To GetRegSubKeyCount(hkey$)
  Write GetRegSubKey(hkey$, i);
  Print ", ";
Next
```

```
CloseRegKey hkey$
```

Remarks

To retrieve the index of the last subkey, **GetRegSubKeyCount** uses the *RegQueryInfoKey* API function.

GetRegSubKey invokes the *RegEnumKeyEx* API function which enumerates subkeys of the specified open registry key. The function retrieves information about one subkey each time it is called. *RegEnumKeyEx* API also retrieves the time it was last modified.

See Also

vbSaveSetting, vbDeleteSetting, vbGetSettingType, vbGetSetting, GetSetting, GetSettingType, SaveSetting, DeleteSetting, CreateRegKey, OpenRegKey, CloseRegKey, GetRegVal, GetRegValName, GetRegValType, GetRegValNameCount, GetRegSubKey, GetRegSubKeyCount

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

StrComp, StrCmp, StrCmpI, LStrCmp and LStrCmpI Functions

Purpose

Compares two strings using either the **Mode Compare** or Windows Regional setting.

Syntax

% = **StrComp**(string1, string2 [, mode])

% = **StrCmp**(string1, string2)
% = **StrCmpI**(string1, string2)

%= **LStrCmp**(string1, string2)
%= **LStrCmpI**(string1, string2)

string1, string2	*: string values*
%, mode	*: integer values*

Description

All these functions compare two strings and return a value accordingly to whether they are greater, lesser or equal to each other.

StrComp returns an integer value to indicate the result of a string comparison. If string1 is less than string2, the return value is -1, greater than string2, the return value is +1, or equal, the return value is zero. The default comparison is according the current **Mode Compare** setting; however, **StrComp** has a third parameter into which a 'one-time only' comparison mode can be entered and this can take any numeric (not string) value Mode Compare can take.

```
Trace StrComp("Hello", "hallo")    // With Current
  Mode Compare Setting
Trace StrComp("Hello", "hallo", 0) // Binary
  Compare
Trace StrComp("Hello", "hallo", 1) // Text Compare
Debug.Show
```

LStrCmp and **LStrCmpI** work in a very similar way to **StrComp** with the same return values, the main difference being that these function will sort according to Windows Regional settings rather than the Mode Compare setting. **LStrCmp** carries out a case-sensitive comparison, while **LStrCmpI** is non case-senstive and converts all letters to lower case before comparing the two strings. **Note:** in reality, on standard Windows settings, both of these functions provide the same result regardless of case; the same seems to be true of their built-in Window API equivalents, **_lstrcmp** and **_lstrcmpi**.

```
Trace LStrCmp("K", "j")            // Returns 1
  but -1 is expected here...
Trace LStrCmpI("K", "j")
Trace _lstrcmp("K", "j")           // ...but it is
  the same with the Windows function as well
Trace _lstrcmpi("K", "j")
Debug.Show
```

Finally, **StrCmp** and **StrCmpI** perform the same task - the comparison made by **StrCmp** being case-sensitive and by **StrCmpI** non case-sensitive - with the main difference being that the result, which uses the current **Mode Compare** setting, is not restricted to -1, 0 or 1, but is the distance in between the first characters which do not match in the compared strings according to the ANSI table.

```
Trace StrCmp("Hello", "hallo")
Trace StrCmpI("Hello", "hallo")
Trace StrCmp("C", "*")
Trace StrCmpI("C", "*")              // Note:
  StrCmpI converts the 'C' to 'c' before the
  comparison
Debug.Show
```

Remarks

These functions perform the same tasks as the <, > and = operators and can, in some instances, be much faster, while in others, not so (GFA Basic uses these functions internally to parse these operators, so speed differences on straight comparisons should be negligible), as shown in the following example:

```
Local n%, r%, r1?, t#
t# = Timer
For n% = 1 To 10000 : r% = StrComp("Hello",
  "hallo") : Next n%
Debug "StrComp time:" & Timer - t#
t# = Timer
For n% = 1 To 10000 : r% = Iif("Hello" > "hallo",
  1, Iif("Hello" < "hallo", -1, 0)) : Next n%
Debug "Iif Comparison time:" & Timer - t#
Debug.Print
t# = Timer
For n% = 1 To 10000 : r1? = (StrComp("Hello",
  "hallo") = 1) : Next n%
Debug "StrComp time:" & Timer - t#
t# = Timer
For n% = 1 To 10000 : r1? = ("Hello" > "hallo") :
  Next n%
```

```
Debug "Straight Comparison time:" & Timer - t#
Debug.Show
```

The first comparison should always be up to twice as fast, whereas the second is sometimes just faster and sometimes just slower.

{Created by Sjouke Hamstra; Last updated: 02/03/2017 by James Gaite}

Space and String[$] Functions

Purpose

Creates a string consisting of a string expression or space repeated a specified number of spaces.

Syntax

$ = **Space**[$](m%)
$ = **String**[$](m%, a$)
$ = **String**[$](m%, n%)

m%, n%	*: integer expression*
a$	*: string*

Description

Each of these functions create a string composed of another string repeated a certain number of times: with **Space**(m%), the result is a string of spaces m% characters long; with **String**(m%,a$) a string composed of a$ repeated m% times; and **String**(m%, n%) results in a string of length m% made up of **Chr**(n%).

Example

```
Debug.Show
Local b$ = "This is", c$ = "GFA"
Trace b$ & Space(10) & c$       // In both
  expressions
```

```
Trace String$(10, 65)                 // the '$ on the
  end
Trace String(5, c$)                   // is optional.
```

Known Issues

If the value of m% in either the **Space** or **String** function is zero or negative, an 'Access-Violation Exception' error can be thrown (this error was fixed in OCX version 2.342 build 1901).
[Reported by Jean-Marie Melanson. 17/02/17]

Remarks

Without the optional **$** character the function still returns a **String** data type and not a **Variant**.

{Created by Sjouke Hamstra; Last updated: 27/01/2019 by James Gaite}

Len Function

Purpose

Determines the length of a character string or the size of a user defined type.

Syntax

Len(a$ | udt)

a$:sexp
udt:user-defined type, udtvar

Description

Determines the number of characters contained within a string expression and returns this value.

For user defined types and their variables, **Len**(Type) and **Len**(var) return the length of the Types and the Type variable respectively.

Example

```
OpenW # 1
Print Len("Hello world")          //prints 11
Print Len(" Hello world ")        //prints 13
```

Remarks

Len returns the wrong value for a string in a **Variant** array.

```
Global Variant x = "abcdefghijklmnopqrstuvwxyz"
Global z(3) As Variant
z(0) = "abcdefghijklmnopqrstuvwxyz"
z(1) = "abc"
Print x `Len(x)        // Is okay because no array
Print z(0)`Len(z(0))  // Shows 16, is 26
Print z(1)`Len(z(1))  // Shows 16, is 3
```

Len acts as **SizeOf** for Variant arrays and will always return 16, the size of a **Variant**.

The address of the UNICODE string in a **Variant** is obtained with:

```
Print "Address Of string data:"`{V:z(1) + 8}
```

The length is placed in the 4 bytes preceding the array of UNICODE bytes and returns the length in bytes. To convert to characters the result must be divided by 2.

```
Global z(3) As Variant
z(1) = "abc"
Print "Len in bytes:"`{{V:z(1) + 8} - 4}
Print "Len in chars:"`{{V:z(1) + 8} - 4} / 2
```

See Also

SizeOf, Variant

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

Lset Command

Purpose

1. Moves a string expression, left justified, to a string.

2. Moves the contents of one type variable to another.

Syntax

Lset a$ = b$

Lset t1 = t2

a$	*: svar, b$: sexp*
t1, t2	*: typevars*

Description

Lset a$=b$ will, first of all, replace all characters in a$ with spaces. Next, b$ is moved into a$ left justified. If b$ contains more characters than a$, then only as many characters as there are "places" for in a$ are moved.

Similar to VB **Lset** also moves the contents of one type variable to another.

Example

Lset with strings.

```
OpenW # 1
Local a$ = String$(15, "-")
Local b$ = "Hello GFA"
```

```
Print a$, Len(a$) // prints --------------- 15
Print b$, Len(b$) // prints HelloGFA        9
Lset a$ = b$
Print a$, Len(a$) // prints Hello GFA       15
```

Lset with a user defined type.

```
Type a : - Long a : End Type
Type b : - Single b : End Type
Local a As a, b As b
b.b = 1.0
Print LPeek(V:b.b), Hex(LPeek(V:b.b), 16)
Lset a = b
Print a.a, Hex(a.a, 16)
```

All bytes of the Type variable b are copied into the Type variable a.

Remarks

Lset for types is similar to a = b, but also works with different Type's. An alternative would be to copy the contents using a memory copy instruction like MemCpy or Bmove. For instance:

```
MemCpy(V:a, V:b, Min(Len(a), Len(b)))
```

There exists no implemented command to copy a String variable and a Type variable together (Lset itself works only with String or only with Type's, not in mixed case). Nevertheless, it's very simple to copy a Type variable to a string:

```
Local t As User_defined_Type
Local a$
a$ = Peek$(V:t, SizeOf(t))
```

or, when a$ has the correct length:

```
BMove V:t, V:a$, Len(a$)
```

To move the string contents back to a type variable use Poke$:

```
Poke$ V:t, SizeOf(t)
```

See Also

Rset, Mid

{Created by Sjouke Hamstra; Last updated: 16/10/2017 by James Gaite}

Rset Command

Purpose

Moves a string expression, right justified, to a string.

Syntax

Rset a$=b$

a$:svar
b$:sexp

Description

Rset a$=b$ will, first of all, replace all characters in a$ with spaces. Next, b$ is moved into a$ right justified. If b$ contains more characters than a$, then only as many characters as there are "places" for in a$ are moved.

Example

```
OpenW # 1
Local a$ = String$(15, "-")
Local b$ = "Hello GFA"
Print a$``Len(a$)       //prints   ---------------
 15
Print b$``Len(b$)       //prints   Hello GFA 9
Rset a$ = b$
Print a$``Len(a$)       //prints   Hello GFA 15
```

See Also

Lset, Mid

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

LTrim, RTrim and Trim Function

Action

Removes spaces at the beginning and/or end of a string expression.

Syntax

Trim[$](x$)

LTrim[$](x$)

RTrim[$](x$)

x$: svar

Description

With the function **LTrim**() you can removeempty spaces from the beginning string, with **RTrim** from the end and with **Trim** from both sides.

Example

```
OpenW 1 : Color , RGB(220, 220, 220)
Local a$ = " GFA", b$ = " Software ", x%
Print a$ + b$
Print LTrim(a$) + RTrim(b$)
Print Trim(a$ & b$)
```

Remarks

See Also

ZTrim

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

ZTrim Function

Purpose

Copies a string until the first null-byte.

Syntax

$ = **ZTrim**[**$**](a$)

a$:sexp

Description

ZTrim(a$) scans the string for the first occurrence of a null-byte, a Chr(0) or #0. It then returns the contents up to the null-byte. When the string doesn't contain a null-byte the entire string is returned.

The function a$ = ZTrim(a$) is similar with

```
If InStr(a$, #0) Then a$ = Left(a$, InStr(a$, #0))
```

and

```
a$ = Char{V:a$}
```

ZTrim most useful with Windows API functions that return strings in a fixed length buffer.

Example

```
Local buf As String*32
Local iLen As Int = 32
~GetComputerName(V:buf, V:iLen)
```

```
buf = ZTrim(buf)
'or:
'buf = Left(buf, iLen)
Message buf, iLen
```

Remarks

See Also

Char{}, LTrim(), RTrim(), Trim()

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Mode Command

Purpose

Sets different options for string conversions and comparisons.

Syntax

Mode [**All** | **BaseYear** | **Compare** | **Date** | **Format** | **Lang** | **StrSpace** | **Using** | **Val**] exp

exp : all variable types

Description

Mode sets a global option for converting values to strings and comparising strings.

- Mode All *svar* Show

- Mode BaseYear *sexp* Show

- Mode Compare *iexp* | *sexp* Show

- Mode Date *sexp* Show

- Mode Format *sexp* Show

- Mode Lang *sexp* Show

- Mode StrSpace *flag* Show

- Mode Using *sexp* Show

- **Mode Val** *sexp* Show

Remarks

The **Mode**() function returns the current settings for the options.

See Also

Mode, Using, Format, Str, Date$, Time$, Val

{Created by Sjouke Hamstra; Last updated: 16/09/2015 by James Gaite}

Xlate$ Function

Purpose

Replaces all characters of a string expression with values from a table.

Syntax

$ = **Xlate**[**$**](a$, mi())

$ = **Xlate**[**$**](a$, m$())

$ = **Xlate**[**$**](a$, addr)

a$:sexp
mi():integer array variable (%,&,|)
m$():string array variable
addr:iexp

Description

Xlate$(a$, m()) converts each character in the string expression a$ using the user-created table in m(). The array can be of any integer type (Byte, Short, Card or Integer/Long).

The character of the string is replaced with value in the array at the index which corresponds with the ASCII code of the character. For instance, when the string a$ contains the character 'A', then it is replaced with the character value at index = 65 in the array, because the ASCII code of 'A' is 65.

XLate() can also take an address of a byte array of 256 characters. This provides a way to use a string as a replacement table. See the example.

GFA-BASIC 32 extends the **XLate** function by using a string array rather than an integer array which only contains one character value. By using a 256 elements string array each character in a$ can be replaced by an entire string, instead of only one character. See example 2.

Example

```
Local a$, b$ , i%
For i% = 0 To 255 ' Create a table
  b$ = Chr$(i%)
  If b$ = Upper$(b$)
    b$ = Lower$(b$)
  Else
    b$ = Upper$(b$)
  EndIf
  a$ = a$ + b$
Next i%
Message XLate$("Hallo World abcABCäöüÄÖÜ", V:a$)
```

prints: hALLO wORLD ABCabcÄÖÜäöü

```
Local a$, b$
InitEscape
Local i%, j%, t#
t = Timer
For i = 1 To 100
  b = XLate(String(500, "This is a test äöüÄÖÜ"),
    htmlEscape())
Next
Print Timer - t, b
Do
  Sleep
```

```
Loop Until Me Is Nothing

Sub InitEscape
  Local out$, i%, c%
  Global Dim htmlEscape$(0 .. 255)
  For c = 0 To 255
    Switch c
    Case ">" : out$ = ">"
    Case "<" : out$ = "<"
    Case "&" : out$ = "&amp;"
    Case "ä" : out$ = "&auml;"
    Case "Ä" : out$ = "&Auml;"
    Case "ö" : out$ = "&ouml;"
    Case "Ö" : out$ = "&Ouml;"
    Case "ü" : out$ = "&uuml;"
    Case "Ü" : out$ = "&Uuml;"
    Case "ß" : out$ = "&szlig;"
    Case 0 To 31, 128 To :
      out$ = "&#" & Dec$(c) & ";"
    Default : out$ = Chr(c)
    EndSelect
    htmlEscape(c) = out
  Next
End Sub
```

Remarks

Xlate$(a$, m%()) corresponds to

```
For i% = 1 To Len(a$)
  Mid$(a$, i%) = Chr(m%(Asc(Mid$(a$, i%, 1))))
Next i%
```

See Also

Upper$(), Lower$(), UCase$(), LCase$()

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

preMatch Function

Purpose

Compiles a regular expression into an internal format.

Syntax

x$ = **preMatch**(pattern$)

x$:svar
pattern$: Regular expression

Description

The **preMatch** function converts *pattern* into an internal format for faster execution. This allows for more efficient use of regular expressions in loops. The string containing the internal format is used as a pattern in **reMatch**, **reSub**, or **Split**. The internal format is identified by four leading bytes "]"#4#2"]".

Example

```
OpenW 1
Local a$, p$
p$ = preMatch(" ?ieter")
While _Data
  Read a$
  If reMatch(a$, p$)
    Print a$
  End If
Wend
```

```
Data
  "Harold","Dieter","Wolfgang","Erhard","Pieter"
```

Remarks

An overview of the regular expression pattern can be found in the topic **reMatch**.

Known Issues

preMatch("?ieter") causes an error as the function does not seem to like the '?' to be the first character; you can get round this by placing a space (which is then ignored) in front of the leading '?' as in the example above.

See Also

reMatch, reSub, Split

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

{}, Byte{}, SByte{}, Word{}, Int{}, Long{}, Large{}, Single{}, Double{}, Short{}, Card{}, UShort{}, Uword{}, Cur{}, Char{} Functions

Purpose

Reads a value from an address.

Syntax

int = **{** address **}**

byte = **Byte{** address **}**

int = **SByte{** address **}**

word = **Word{** address **}**

integer = **Int{** address **}**

long = **Long{** address **}**

large = **Large{** address **}**

single = **Single{** address **}**

double = **Double{** address **}**

short = **Short{** address **}**

card = **Card{** address**}**

card = **Ushort{** address**}**

card = **Uword{** address **}**

currency = **Cur{** address **}**

string = **Char{**address**}**

address:address

Description

Reads the specified data type from address.

Example

```
OpenW # 1
Dim a As Double = 1.2345, x%, i%
Print Hex$({*a}, 8)``Hex$({*a + 4}, 8)
Print
For i% = 0 To 7
  Print Hex$(Peek(*a + i%), 2);
Next i%
Print : Print a
```

Prints first 126E978D 3FF3C083, which is the internal representation of variable a as a long word and then 8D976E1283C0F33F, which is the internal representation of a read in as bytes.

See Also

Peek() Functions

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

{}= Command

Purpose

Writes a value in a specified data type to an address.

Syntax

{addr**}** = exp

Byte{ address **} =** exp

SByte{ address **} =** exp

Word{ address **} =** exp

Int{ address **} =** exp

Long{ address **} =** exp

Large{ address **} =** exp

Single{ address **} =** exp

Double{ address **} =** exp

Short{ address **} =** exp

Card{ address**} =** exp

Ushort{ address**} =** exp

Uword{ address **} =** exp

Cur{ address **} =** exp

Char{address**} =** exp

address:address
expaexp

Description

Writes a value in the specified data type to an address.

{}= writes a 32-bit value.

Example

```
Dim a% = 5
{*a%} = Int{*a%} + 1 // a slow a%++
Print a%
```

And...

```
Dim a@, b@, c@, f@
a = 2222222222.56
b = 1111111111.66
c = Cur{V:a} + Cur{V:b}
Print c
Cur{V:f}= 65000
Print Cur{V:f} // reads the buffer
```

Remarks

The **{}=** commands have corresponding **Poke** commands, which can be used instead.

See Also

Poke Commands

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Let Command

Purpose

Assignment of variables

Syntax

Let x=y

x:avar or svar
y:aexp or sexp

Description

Let x = y command assigns the variable x with the value in expression y. x and y must either be both numeric or both strings.

Let x=y is normally not necessary, but is used when one of the reserved GFA-BASIC variables (for example Data) needs to be assigned a value.

Example

```
Local data As String
Let data = Str$(PI, 9)
```

See Also

Lset

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

:=, = Assignment operators

Purpose

Assigns an expression or value to a variable.

Syntax

varname **:=** value

varname **=** value

Description

Assignment operator **:=** is often used with assignment of arguments of OLE object properties, this is called *passing named arguments*. Using named arguments are provided as a shortcut for typing argument values. With named arguments, you can provide any or all of the arguments, in any order, by assigning a value to the named argument. You do this by typing the argument name plus a colon followed by an equal sign and the value (Argument := Value) and placing that assignment in any sequence delimited by commas.

Example

```
Local a$
a$ := "Hello"
a$ = "Hello"
```

Named arguments with objects. Notice that the arguments in the following example are in the reverse order of the expected arguments:

```
// Raises the error: "Put #/Get # without field
  and without variable"
Err.Raise Description := "", Number := 71
```

See Also

+=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 18/09/2014 by James Gaite}

Clr Command

Purpose

Deletes all variables listed after this command.

Syntax

Clr x1[,x2,...]

x1,x2,...:variables of any type

Description

The variables in the list to be deleted with **Clr** must be separated by commas. The arrays cannot be deleted using **Clr**.

Example

```
Dim a$ = "Init"
Dim ar$(100)
Print a$        // "Init"
Clr a$
Erase ar$()
Print a$        // ""
```

See Also

Clear, Erase

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Global Command

Purpose

Used to declare global variables and allocate storage space.

Syntax

Global [**Dim**] varname[()] [**As** [**New**] type] [= value], …

Global type varname1 [= value], varname2 [= value], …

Global varname1**$** [= value], varname2**%** [= value], …

varname: name of variable

*type: Optional. Data type of the variable; may be **Byte**, **Boolean**, **Card**, **Short**, **Word**, **Integer**, **Long**, **Large**, **Currency**, **Single**, **Double**, **Date**, **String**, (for variable-length strings), **String** * length (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. Use a separate **As** type clause for each variable being defined.*

Description

The **New** keyword enables implicit creation of a few GFA-BASIC 32 objects, like **DisAsm**, **Collection**, **StdFont, Font**, **StdPicture, Picture, CommDlg,** and **ImageList**. If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it, so you don't have to use the **Set** statement to assign the object reference. The **New** keyword can't be used to declare variables of any intrinsic data type.

Variables declared using the **Global** (or **Public)** statement are available to all procedures in the program.

If you don't specify a data type or object type and there is no **Def***type* statement in the module, the variable is **Variant** by default.

Variables can be initialized while they are declared.

When a variable isn't explicitly initialized, a numeric variable is initialized to 0, a variable-length string is initialized to a zero-length string (""), and a fixed-length string is filled with zeros. **Variant** variables are initialized to Empty. Each element of a user-defined type variable is initialized as if it were a separate variable.

Example

```
Global a As Int, b%, d As Handle, e$
Global Double a, b, c, d, e
Dim a As Int, b%
Global Dim a(100) As String
Global a(100) As String
Dim a$(100)
Global dis As New DisAsm
Dim col As New Collection
```

Remarks

If you use **Dim** in the main part of a program, the variables will be declared Global. When Dim is used in a sub the variables are local.

See Also

Dim, Local, Static

Boolean, Byte, Card, Short, Word, Int16, Long, Int, Integer, Int32, Int64, Large, Single, Double, Currency, Date, Handle, String, Variant, Object

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Local Command

Purpose

Declares local variables in a subroutine or main program.

Syntax

Local [**Dim**] varname[()] [**As** [**New**] type] [= value], ...

Local type varname1 [= value], varname2 [= value], ...

Local varname1**$** [= value], varname2**%** [= value], ...

varname: name of variable

*type: Optional. Data type of the variable; may be **Byte**, **Boolean**, **Card**, **Short**, **Word**, **Integer**, **Long**, **Large**, **Currency**, **Single**, **Double**, **Date**, **String**, (for variable-length strings), **String** * length (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. Use a separate **As** type clause for each variable being defined.*

Description

Local declares local variables. When used in the main program, the variable's scope is limited to the main part and isn't known in subroutines. In this respect, **Dim** and **Local** differ. **Dim** declares local variables as well, but when declared in the main program they are considered global.

The **New** keyword enables implicit creation of a few GFA-BASIC 32 objects, like **DisAsm**, **Collection**, **StdFont,**

Font, **StdPicture, Picture, CommDlg,** and **ImageList**. If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it, so you don't have to use the **Set** statement to assign the object reference. The **New** keyword can't be used to declare variables of any intrinsic data type.

Variables declared using the **Global** (or **Public)** statement are available to all procedures in the program.

If you don't specify a data type or object type and there is no **Def***type* statement in the module, the variable is **Variant** by default.

Variables can be initialized while they are declared.

When a variable isn't explicitly initialized, a numeric variable is initialized to 0, a variable-length string is initialized to a zero-length string (""), and a fixed-length string is filled with zeros. **Variant** variables are initialized to Empty. Each element of a user-defined type variable is initialized as if it were a separate variable.

Example

```
OpenW 1
AutoRedraw = 1
Global a%, x%, i%
a% = 0
For i% = 1 To 10
  a% += i%
  test (a%)
Next i%
Print a% // Prints; 205

Procedure test(ByRef a%)
  Local i%
  For i% = 1 To 5
    a% += i%
  Next i%
EndProc
```

The For...Next loop counter i% is defined both as a global and a local variable.

Remarks

See Global for a more detailed description.

Known Issues

When using local arrays, you may get a memory leak problem. This stems from the fact that the compiler forgets to add destruction code for local arrays when an explicit local declaration of a string variable is absent. As a workaround, in any procedure, function or sub which declares a local array, add a local string variable dummy$ if none other is present.

See Also

Global, Dim, Static

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Using Data Statements

Purpose

To populate variables and arrays with pre-defined constants.

Syntax

Restore [label]
Read k1[,k2,k3,...]
Data k1[,k2,k3,...]

_Data = x%
x% = **_Data**

label	*: user defined label*
k1, k2,k3...	*: numerical and/or string literals*
x%	*: integer*

Description

By using Data statements, it is possible to read in pre-defined numerical, date and string values into variables or arrays in an economical fashion.

The data itself is stored in statements prefaced with the **Data** command, and this is read into the desired variables by using the **Read** command, as shown below:

```
Local a%(2)
Read a%(0), a%(1), a%(2)
Print a%(0), a%(1), a%(2)
Data 1,2,3
```

Data is added to a data statement separated by commas, and different types of variable types are added in the following way:

- Numbers - in plain form i.e. 1, 2, 3.01, etc; also recognised literals can used, such as %1010 for binary, $FF for hexadecimal, etc.
- Strings - in plain form or inside inverted commas (strings which contain a comma should be in inverted commas).
- Dates - in the form "#dd.mm.yyyy#" or "#mm/dd/yyyy#".
- Null - as "#Null#" (this allows the initialisation of variants).

Hence, a data statement could look like this:

```
Data 1,1.456,%1010,$FF,&O12,String,"Another
  string","String, with a
  comma","#09/10/1980#","#Null#"
// Integer, Decimal, Integer in Binary, ...in Hex,
  ...in Octal, String, String, String, Date, Null
```

When a program is run which contains a data statement, an internal data pointer is set to the first item of data within the whole program. When that item of data is read from a **Data** statement, the internal data counter is incremented so that the **Read** command knows the position of the next data item to be read in. The value (or position) of this data pointer can be retrieved using the **_Data** function like so:

```
Local a$(2), n%
Print , _Data   // Prints the initial position of
  the data pointer
For n% = 0 To 2
  Read a$(n%)
  Print a$(n%), // Shows the read data...
  Print _Data   // ...and the position of the next
    item of data
Next n%
Data "Record 1","Record 2","Record 3"
```

_Data can also be used as a command to set the position of the data pointer like this:

```
Dim a%, dp%(10), n%
For n% = 1 To 10
  dp%(n%) = _Data       // Store the data pointer
  Read a%               // Use Read to move the
    pointer along
Next n%
For n% = 10 DownTo 1     // Run backwards through
  dp%()...
  _Data = dp%(n%)       // ...and set the pointer
    so that...
  Read a% : Print a%    // ...the data is read
    backwards
Next n%
Data 1,2,3,4,5,6,7,8,9,10
```

Generally, data in statements will be grouped together in memory and so, if you know the start position of the first item, you can work out the position of others. To illustrate this: in the last example, all the data was an integer lower than 65536 and thus was stored as a 16-bit integer. With this knowledge, the above example could be shortened to this:

```
Dim a%, dp%, n%
// Set dp% to the last 16-bit integer which means
  moving...
// ... past 9 other 16-bit or 2-byte values
dp% = _Data + (9 * 2)
```

```
// Now read through the data, decreasing the data
  pointer by...
// ...2 for each 16-bit integer
For n% = 1 To 10
  _Data = dp% : Sub dp%, 2
  Read a%  : Print a%
Next n%
Data 1,2,3,4,5,6,7,8,9,10
```

When the **Read** command has read the last data item in a program, then **_Data** is set to zero. In this way, it is possible to determine whether the last item has been read, as in the next example:

```
Local a As Variant
While _Data
  Read a : Print a
Wend
Data 1,1.456,%1010,$FF,&O12,String,"Another
  string"
Data "String, with a
  comma","#09/10/1980#","#Null#"
```

Where there are numerous different blocks of data to be read in, it is possible that the data to be passed to a specific variable or array is not the first in the list. In this case, the data pointer can be repositioned using the **Restore** command. If used with a label, then the data pointer is moved to first data item after that label (the label must be in the same procedure as the **Restore** command); however, if no label is used, the pointer is moved back to the first item of data in the program listing.

```
Local a(3) As Variant, b$(3), n%
Restore variants         // Jump to the last data
  group to read in variant values
```

```
For n = 0 To 3 : Read a(n%) : Print a(n%) : Next
  n%
Restore                      // Place data pointer back
  at start to read strings.
For n = 0 To 2 : Read b$(n%) : Print b$(n%) : Next
  n%
strings:
Data "String1",String2,String3
integers:
Data 1,2,3,4,5
variants:
Data "#Null#",String,5,5.6
```

NOTE: Data statements are NOT procedure-specific but can be read from anywhere within the program; hence if you Read more items than are in the Data statements in the current procedure, the data pointer will move to the next Data statement it finds and continue to read from there until there are none further, when an error will be thrown. In addition, if a **Restore** command is not used, the data pointer starts from the first data statement it finds in the program listing, regardless of whether it is in the current procedure. This behaviour is possibly a throwback and/or compatibility measure to when procedures where not so 'stand alone'. The following examples illustrates this behaviour:

```
DataRead2 // Reads 8 items of data from DataRead1
DataRead1 // Reads the remaining 2 items of data
  from DataRead1 and then the first 6 from
  DataRead2

Procedure DataRead1
  Local n%, a$
  For n% = 1 To 8 : Read a$ : Print a$; " "; : Next
    n% : Print
  Data A,B,C,D,E,F,G,H,I,J
```

```
EndProcedure

Procedure DataRead2
  Restore
  Local n%, a$
  For n% = 1 To 8 : Read a$ : Print a$; " "; : Next
    n% : Print
  Data K,L,M,N,O,P,Q,R,S,T
EndProcedure
```

Remarks

1. Another way to initialize an array is by using the Array()= command. This command doesn't require Data lines, but instead assigns data as part of a string.

2. **Read** and **Data** statements can not be used in lg32 Libraries.

{Created by Sjouke Hamstra; Last updated: 17/05/2017 by James Gaite}

DefType Statements

Purpose

set the default data type for variables and arguments.

Syntax

Def*Type* letterrange$[, letterrange$] . . .

letterrange$: "letter1[-letter2]"

Description

The **Def***Type* commands simplify variable declaration. The letter1 and letter2 arguments specify the name range for which you can set a default data type. Each argument represents the first letter of the variable, argument, Function procedure, or Property Get procedure name and can be any letter of the alphabet. The case of letters in letterrange$ isn't significant.

letterrange$	Variables that start with
"b"	'b' (or 'B')
"bo"	'b' **or** 'o'
"x-z"	'x', 'y' or 'z'
"b-d,x-z"	'b' to 'd' and 'x' to 'z'.

The statement name determines the data type:

Statement	Data Type
DefBool, DefBit	Boolean
DefByte	Byte

DefCrd	Card
DefInt16, DefWrd	16-bit Integer
DefInt, DefInt32	32-bit Integer
DefLng	Long
DefLar, DefInt64	Large integer (64-bit) (synonym:)
DefCur	Currency
DefSng, DefFlt	Single
DefDbl	Double
DefDate	Date
DefStr	String
DefVar	Variant

Example

```
DefCrd "bo"
Dim b = 2
Print TypeName(b) // Card
```

Remarks

Once the range A-Z has been specified, you can't further redefine any sub ranges of variables using Deftype statements. Once a range has been specified, if you include a previously defined letter in another Deftype statement, an error occurs. However, you can explicitly specify the data type of any variable, defined or not, using a Dim statement with an As type clause. For example, you can use the following code at module level to define a variable as a Double even though the default data type is Integer:

```
DefInt "A-Z"
Dim TaxRate As Double
```

Def*type* statements don't affect elements of user-defined types because the elements must be explicitly declared. Variable types can also be declared by appending the relevant postfix characters !, |, &, %, # or $.

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

ProcAddr Function

Purpose

Returns the memory address of a subroutine.

Syntax

a% = **ProcAddr**(procname)

a%	*: iexp*
procname	*: proceedure name*

Description

With the function **ProcAddr**() you determine the address of a **Sub**, **Procedure**, **Function**, or **FunctionVar**. The return is an Integer value.

ProcAddr permits the address of the procedure to be passed to a Windows API function in a dynamic-link library (DLL), rather passing the procedure's return value. The API function can then use the address to call the Basic procedure, a process known as a callback.

For example, the *EnumWindows* function from the Win32 API (built-in)

Function EnumWindows(lpEnumFunc as Long, lParam as Long) As Long

EnumWindows is an enumeration function, which means that it can list the handle of every open window on your system. *EnumWindows* works by repeatedly calling the function you pass to its first argument (lpEnumFunc). Each time *EnumWindows* calls the function, *EnumWindows* passes it the handle of an open window.

When you call *EnumWindows* from your code, you pass a user-defined function to this first argument to handle the stream of values. For example, you might write a function to add the values to a list box, convert the hWnd values to window names, or take whatever action you choose.

To specify that you're passing a user-defined function as an argument, you first obtain the address of the function with the **ProcAddr**, and then pass that address to the first parameter of *EnumWindows*. Any suitable value can be passed to the second argument. The user-defined function you specify when you call the procedure is referred to as the *callback function*. Callback functions (or "callbacks," as they are commonly called) can perform any action you specify with the data supplied by the procedure. See example.

A callback function must have a specific set of arguments, as determined by the API from which the callback is referenced. Refer to your API documentation for information on the necessary arguments and how to call them.

In the same way the ProcAddr function can be used to obtain the address of a window function when registering a window class. Hook function can be implemented as anything that requires a function pointer.

Example

```
OpenW 1, , , 300, 300
Ocx ListBox lb = , 10, 10, 280, 200
Dim cnt%
```

```
Trace EnumWindows(ProcAddr(EnumWndProc), V:cnt)
MsgBox "Window count: " & Dec(cnt)
Do
  Sleep
Until Me Is Nothing

Function EnumWndProc(hWnd As Long, lParam As Long)
  As Long
  ' Increment count
  {lParam} = {lParam} + 1
  ' Get window title and insert into ListBox
  Dim s As String = _Win$(hWnd)
  If s  Then
    lb.AddItem s
    lb.ItemData(lb.NewIndex) = hWnd
  End If
  EnumWndProc = True    '  keep enumerating
End Function
```

Remarks

You can use **ProcAddr** to call a function or procedure through such a pointer from within Basic using the **StdCall**(**ProcAddr**(subname))().

See Also

LabelAddr, DisAsm, StdCall

{Created by Sjouke Hamstra; Last updated: 17/05/2017 by James Gaite}

LabelAddr Function

Purpose

Returns the memory address of a label.

Syntax

a% = **LabelAddr**(name)

a%:iexp
name:label name

Description

With the function **LabelAddr**(name) you determine the address of a named label. The return is an Integer value.

A label can be number or alphanumeric name followed by a semicolon.

Example

```
OpenW 1
Print LabelAddr(test)
Print LabelAddr(5)
5            // numeric label (without :)
'
test:        // alphnumeric label (with :)
```

Remarks

The address of a label can be obtained for a label that is in scope. Labels can be used locally only.

See Also

ProcAddr, DisAsm

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

SizeOf Function

Purpose

Returns the size of a variable or a user-defined type.

Syntax

SizeOf(a)

a:variable or user-defined type

Description

SizeOf returns the number of bytes a variable or user-defined type occupies.

String variables always return 4, and **Variant** variables always return 16.

For an array only the size of an element can be determined.

Example

Strings

```
OpenW 1
Local a$, x%, b As String*10000
a$ = Space$(1000)
Print SizeOf(a$)      // prints 4
Print SizeOf(b)       // prints 10000
Print Len(a$), Len(b) // prints 1000, 10000
```

Type variables

```
OpenW 1
Type test // Packed 1
  - String*10 a$
  - Byte b(5)
  - String*5 c(5)
  - UByte d(5)
  - Variant e(5)
  - Double f(5)
  - Word g(5)
  - Currency h(5)
  - Large j(5)
  - Int k(5)
  - Float l(5)
  - String*10 m$
EndType
Global R As test
Print "b()-Byte", SizeOf(R.b(1))
Print "c()-String*5", SizeOf(R.c(1))
Print "d()-UByte", SizeOf(R.d(1))
Print "j()-Large", SizeOf(R.j(1))
Print "k()-Int", SizeOf(R.k(1))
Print "l()-Float", SizeOf(R.l(1))
Print "m-String*10", SizeOf(R.m$)
```

Remarks

SizeOf is compatible with C.

See Also

BitSizeOf, Len

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

BitSizeOf Function

Purpose

Returns the size of a fixed or a numeric variable in bits

Syntax

% = **BitSizeOf**(variable)

variable: avar

Description

With the function **BitSizeOf** you get the size of a numeric variable, a fixed string, type elements, array’s, etc..

Example

```
OpenW 1
Type atest
  - Byte a
  - Int b
  - Double c
  - Byte d
EndType
Dim a As atest
a.a = 1 : a.b = 2 : a.c = 3 : a.d = 4
Print " Size of the element in bit"
Print "BitSizeOf of a: ", BitSizeOf(a.a)
Print "BitSizeOf of b: ", BitSizeOf(a.b)
Print "BitSizeOf of c: ", BitSizeOf(a.c)
Print "BitSizeOf of d: ", BitSizeOf(a.d)
Print
```

```
Print " Size of the element's in byte"
Print "SizeOf of a: ", SizeOf(a.a)
Print "SizeOf of b: ", SizeOf(a.b)
Print "SizeOf of c: ", SizeOf(a.c)
Print "SizeOf of d: ", SizeOf(a.d)
Do
  Sleep
Until Me Is Nothing
```

Remarks

See Also

SizeOf, BitOffsetOf, OffsetOf

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

BitOffsetOf, OffSetOf Function

Purpose

Retrieves the offset of a member from the beginning of its parent structure.

Syntax

% = **BitOffsetOf**(Type.Member)

% = **OffsetOf**(Type.Member)

Description

BitOffsetOf and **OffsetOf** return the start position of an element of a Type by using its name in bits and bytes respectively.

Example

```
OpenW  # 1,  10, 10, 300, 450, $030
// $030 => Window with caption & close box
TitleW # 1, "Demo BitOffsetOf()"
Type atest
  - Byte a
  - Int b
  - Double c
  - Byte d
EndType
Dim a As atest
a.a = 1 : a.b = 2
```

```
a.c = 3 : a.d = 4
Print " Which element's are used?"
Print "element a = Byte : ", a.a
Print "element b = Int : ", a.b
Print "element c = Double: ", a.c
Print "element d = Byte : ", a.d
Print " start address of an element"
Print "start address ->Type: ", V:a
Print "start address a: ", V:a.a
Print "start address b: ", V:a.b
Print "start address c: ", V:a.c
Print "start address d: ", V:a.d
Print " Where begins an element in" " the Type in
  byte?"
Print "OffSetOf of a: ", Space$(20), OffsetOf(a.a)
Print "OffSetOf of b: ", Space$(20), OffsetOf(a.b)
Print "OffSetOf of c: ", Space$(20), OffsetOf(a.c)
Print "OffSetOf of d: ", Space$(20), OffsetOf(a.d)
Print " Start of type elements in bit:"
Print "BitOffsetOf of a: ", BitOffsetOf(a.a)
Print "BitOffsetOf of b: ", BitOffsetOf(a.b)
Print "BitOffsetOf of c: ", BitOffsetOf(a.c)
Print "BitOffsetOf of d: ", BitOffsetOf(a.d)
Print " Size of the element's in byte"
Print "SizeOf of a: ", Space$(20), SizeOf(a.a)
Print "SizeOf of b: ", Space$(20), SizeOf(a.b)
Print "SizeOf of c: ", Space$(20), SizeOf(a.c)
Print "SizeOf of d: ", Space$(20), SizeOf(a.d)
Print " Size of the element's in bit"
Print "BitSizeOf of a: ", BitSizeOf(a.a)
Print "BitSizeOf of b: ", BitSizeOf(a.b)
Print "BitSizeOf of c: ", BitSizeOf(a.c)
Print "BitSizeOf of d: ", BitSizeOf(a.d)
Do
  Sleep
Until Me Is Nothing
CloseW 1
```

Remarks

Also it’s possible to determine the **BitOffsetOf** an element of an array, or of an array in a Type or of a Type in Type and also of a Type in an array.

See Also

BitSizeOf, SizeOf, V:

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Control Command

Purpose

Creates a control in the current active form, window, or dialog.

Syntax

Control text$, id%, class$, x, y, w, h[, style%]

text$:control text
id%:control identifier
class$:class name
x,y,w,h:iexp
style%:the control styles

Description

Control creates a program defined control window with width w% and height h% at coordinates specified in x% and y%. The window shows the text specified in text$ and can be referred to with the value specified in ID%. class$ specifies the class of the control elements which the control window can assign.

The command creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND and WM_NOTIFY messages should be handled in the form's **_MessageProc** sub.

Example

```
/* Styles for the UpDown Control
Global Enum UDS_WRAP = 1, _
  UDS_SETBUDDYINT, UDS_ALIGNRIGHT=4, _
  UDS_ALIGNLEFT=8, UDS_AUTOBUDDY=10, _
  UDS_ARROWKEYS=$20, UDS_HORZ =$40, _
  UDS_NOTHOUSANDS=$80, UDS_HOTTRACK =$100
/* Messages to Control the animation
Global Enum UDM_SETRANGE=WM_USER + 101, _
  UDM_GETRANGE, UDM_SETPOS, UDM_GETPOS, _
  UDM_SETBUDDY, UDM_GETBUDDY, UDM_SETACCEL, _
  UDM_GETACCEL, UDM_SETBASE, UDM_GETBASE, _
  UDM_SETRANGE32, UDM_GETRANGE32, _
  UDM_SETUNICODEFORMAT=$2005, _
  UDM_GETUNICODEFORMAT=$2006
OpenW 1
Ocx TextBox ed1 = "", 10, 10, 100, 20
ed1.Appearance = 1
Control "", 1010, "msctls_updown32", _
  UDS_ARROWKEYS | UDS_WRAP | UDS_SETBUDDYINT |
    UDS_ALIGNLEFT | _
  WS_TABSTOP,  10, 10, 100, 20
Local hUpDown As Handle =  Dlg(Win_1.hWnd, 1010)
SendMessage hUpDown, UDM_SETBUDDY, ed1.hWnd, 0
SendMessage hUpDown, UDM_SETRANGE, 0,
  MakeLong(1000, 990)
SendMessage hUpDown, UDM_SETPOS, 0, MakeLong(0,
  993)
~SetFocus(Dlg(Win_1.hWnd, 10))
Do
  Sleep
Until Me Is Nothing
```

Remarks

With the general **Control** statement any control type can be created.

See Also

AutoCheckBox, AnimateCtrl, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

CheckBox Control

Purpose

Creates a checkbox.

Syntax

CheckBox text$, ID%, x%, y%, w%, h%[,style%]

Description

A **CheckBox** is a rectangle which has diagonals drawn in it when clicked on with the mouse. The text specified in text$ is displayed right justified next to the rectangle. A CheckBox can contain the WS_TABSTOP and WS_GROUP style elements.

Messages from the CheckBox are handled in the **_Message** event sub of the parent

Example

```
Dlg 3D On
Local style1%, style2%, x%
style1% = BS_AUTOCHECKBOX | WS_TABSTOP | WS_BORDER
style2% = BS_GROUPBOX | WS_GROUP
Dialog # 1, 10, 10, 310, 170, "Dialog 2",
  WS_SYSMENU
  CheckBox "Check1", 11, 50, 50, 80, 30, style1%
  CheckBox "Check2", 12, 50, 100, 80, 30, style1%
  GroupBox "Test field1", 13, 10, 10, 140, 130,
    style2%
  AutoCheckBox "Check3", 13, 170, 50, 80, 30,
    style1%
  AutoCheckBox "Check4", 14, 170, 100, 80, 30,
    style1%
  GroupBox "Test field2", 23, 160, 10, 140, 130,
    style2%
EndDialog
SetCheck 1, 11, 1
SetCheck 1, 14, 1
ShowDialog # 1
Do
  Sleep
Until Me Is Nothing
Dlg 3D Off

Sub Dlg_1_Close(Cancel?)
  Cancel? = False ' don't cancel close
EndSub

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  Switch Mess
  Case WM_COMMAND
    Trace wParam
    If wParam >= 11 And wParam <= 14
      If Check?(1, wParam) Then Message "Checkbox" &
        wParam - 10 & " Checked"
      If Check?(1, wParam) = 0 Then Message
        "Checkbox" & wParam - 10 & " Unchecked"
    EndIf
  EndSwitch
EndSub
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

ComboBox Control

Purpose

Creates a **ComboBox** control.

Syntax

ComboBox ID%, x%, y%, w%, h%[,style%]

Description

A ComboBox is a combination of a ListBox with a static text field or with an EditText element. If the ComboBox contains a static text field, the selected entry from the ListBox is shown in it. Additional style elements for a ComboBox are WS_TABSTOP, WS_GROUP, WS_VSCROLL and WS_DISABLED.

style%:

CBS_SIMPLE ($0001) - specifies a ListBox, which is always shown.

CBS_DROPDOWN ($0002) - similar to CBS_SIMPLE. However, the ListBox is only displayed when the user performs the relevant selection (for example open or change).

CBS_DROPDOWNLIST ($0003) - similar to CBS_DROPDOWN. The difference is that the selection window (edit control) contains the predefined text until the user makes the selection.

CBS_OWNERDRAWFIXED ($0010) - specifies a ListBox, whose input must be performed by the calling task. All items within this ListBox have the same height.

CBS_OWNERDRAWVARIABLE ($0020) - similar to CBS_OWNERDRAWFIXED, except that the items can here have a variable height.

CBS_AUTOHSCROLL ($0040) - when the user enters text which goes beyond the windows or editfield edge the text is automatically scrolled within the output window (edit control).

CBS_OEMCONVERT ($0080) - converts characters from ANSI into OEM and back (for example using your own character table).

CBS_SORT ($0100) - automatically sorts all inputs in a ListBox.

CBS_HASSTRINGS ($0200) - used when items within a ComboBox are composed of strings. The ComboBox refers items to strings by using pointers.

Example

```
Local i%, sel$
Debug .Show
Dlg 3D On : Local x%
Dialog # 1, 20, 20, 300, 200, "Test"
  ComboBox "Combobox", 20, 40, 50, 200, 120,
    CBS_DROPDOWN | _
    CBS_SORT | 2048 | WS_TABSTOP
  PushButton "OK", IDOK, 60, 140, 80, 25,
    BS_DEFPUSHBUTTON | WS_TABSTOP
  PushButton "CANCEL", IDCANCEL, 150, 140, 64, 24,
    BS_DEFPUSHBUTTON |  WS_TABSTOP
```

```
EndDialog
Data "GFA-BASIC 32"
Data "GFA-BASIC for MS-DOS"
Data "GFA-BASIC for Windows"
Data "GFA-BASIC for Atari"
Data "GFA-BASIC for Amiga"
Data ""
For i% = 0 To 5
  Read sel$
  ~SendMessage(Dlg(1, 20), CB_ADDSTRING, 0, sel$)
  _Win$(DlgItem(1, 20)) = sel$
Next
ShowDialog # 1
Do
  Sleep
Until Me Is Nothing
Dlg 3D Off

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  Switch Mess
  Case WM_COMMAND
    Switch LoWord(wParam)
    Case 20
      Trace "Notification code: " & HiWord(wParam)
    Case IDOK, IDCANCEL
      CloseDialog # 1
    EndSwitch
  EndSwitch
EndSub
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general Control statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl, Ocx

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Groupbox Control

Purpose

Creates a group box with the defined text and coordinates.

Syntax

GroupBox text$, ID%,x%,y%,w%,h%[,style%]

Description

A **GroupBox** is a rectangle which can contain several additional control elements (for example Radio buttons). The text specified in text$ is shown in the upper left corner of the Groupbox. A **GroupBox** can contain the WS_TABSTOP and WS_DISABLED style elements.

Example

```
// only a module to show how..
Local style1%, style2%, style3%, style4%
Local style5%, style6%, x%
style1% = WS_TABSTOP
style2% = BS_DEFPUSHBUTTON | WS_TABSTOP
style3% = BS_GROUPBOX | WS_TABSTOP
style4% = BS_AUTORADIOBUTTON | WS_TABSTOP
style5% = BS_AUTOCHECKBOX | WS_TABSTOP
style6% = ES_UPPERCASE | WS_BORDER | _
  WS_TABSTOP
Dlg 3D On
DlgBase Unit
// in Unit (1/4 sign width, 1/8 Zeichen height
Dialog # 1, 50, 50, 300, 200, "Dies ist ein Test"
```

```
  PushButton "Pushbutton 1", 100, 12, 14, 72, 14,
    style1%
  PushButton "Pushbutton 2", 101, 12, 32, 72, 14,
    style1%
  PushButton "Pushbutton 3", 102, 12, 50, 72, 14,
    style1%
  DefPushButton "DefPushbutton", IDOK, 12, 68, 72,
    14, style2%
  GroupBox "Radiobuttons", 106, 89, 14, 56, 53,
    style3%
  RadioButton "Radio 1", 107, 93, 25, 39, 12,
    style4%
  RadioButton "Radio 2", 108, 93, 36, 39, 12,
    style4%
  RadioButton "Radio 3", 109, 93, 47, 39, 12,
    style4%
  CheckBox "Checkbox 1", 110, 17, 94, 61, 12,
    style5%
  CheckBox "Checkbox 2", 111, 17, 107, 61, 12,
    style5%
  CheckBox "AutoCheckbox", 112, 17, 120, 61, 12,
    style5%
  EditText "", 113, 89, 94, 59, 12, style6%
EndDialog
SetCheck 1, 109, 1
SetCheck 1, 112, 1
ShowDialog # 1
Do
  Sleep
Until Me Is Nothing

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  If Mess = WM_COMMAND
    Switch wParam
    Case 100, 101, 102
      _Win$(Dlg(1, 113)) = "Pushbutton " &
        Str(wParam)
    Case IDOK
      CloseDialog # 1
    Case 107, 108, 109
      _Win$(Dlg(1, 113)) = "Radiobutton " &
        Str(wParam)
    Case 110, 111, 112
      _Win$(Dlg(1, 113)) = "Checkbox " & Str(wParam)
    EndSwitch
  EndIf
EndSub
```

Remarks

The preferred way to implement a user interface is by using OCX controls rather than standard window controls. Replace **GroupBox** with the **Frame** OCX control. However, Ocx controls cannot be used in editor extension Dialog boxes.

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

ListBox Control

Purpose

Creates a ListBox control in the current active form, window, or dialog.

Syntax

ListBox text$, id%, x, y, w, h[, style%]

text$:control text
id%:control identifier
x, y, w, h:iexp
style%:the control styles

Description

The control is a rectangle containing a list of strings (such as filenames) from which the user can select.

Style	***Meaning***
LBS_NOTIFY ($0001)	sends a message to the parent window when the user selects an entry by clicking.
LBS_SORT ($0002)	performs an alphabetical sort of several alternatives.
LBS_NOREDRAW ($0004)	prevents ListBox redraw after receiving changes.
LBS_MULTIPLESEL($0008)	after an initial selection (the entry is displayed in reverse) enables additional selections. The

	number of selections is not limited.
LBS_OWNERDRAWFIXED($0010)	specifies a ListBox, whose input must be performed by the calling task. All items within this ListBox have the same height.
LBS_OWNERDRAWVARIABLE ($0020)	similar to LBS_OWNERDRAWFIXED, except that the items can have a variable height.
LBS_HASSTRINGS ($0040)	used when items within a ListBox are composed of strings. The ListBox refers items to strings by using pointers.
LBS_USETABSTOPS ($0080)	displays a multi-column ListBox, whereby the individual columns are located at predefined tab positions.
LBS_NOINTEGRALHEIGHT($0100)	makes the size of the ListBox the size specified by application.
LBS_MULTICOLUMN($0200)	specifies a multi-line ListBox, which can scroll horizontally.
LBS_WANTKEYBOARDINPUT($0400)	allows for assignment of special keys or key combinations (Hotkeys) to entries in a Listbox.
LBS_EXTENDEDSEL($0800)	specifies a ListBox, whereby multiple entries can be selected by using the Shift key and mouse clicks.

LBS_STANDARD	creates a ListBox with the following attributes: LBS_NOTIFY | LBS_SORT | WS_VSCROLL | WS_BORDER

If you do not specify a style, the default style is LBS_NOTIFY | WS_BORDER.

The command creates a control without an OCX wrapper; so it and cannot be handled using properties, methods, and event subs. When used in a form the WM_COMMAND message should be handled in the form's _**Message** sub.

Example

```
Dlg 3D On
Local x%, sel$
Dialog # 1, 10, 10, 200, 310, "Test"
  ListBox "Listbox", 10, 20, 20, 150, 200
  DefPushButton "OK", IDOK, 10, 250, 80, 25, WS_TABSTOP
  PushButton "CANCEL", IDCANCEL, 110, 250, 64, 24,
    WS_TABSTOP
EndDialog
Data "GFA-BASIC 32"
Data "GFA-BASIC for MS-DOS"
Data "GFA-BASIC for Windows"
Data "GFA-BASIC for Atari"
Data "GFA-BASIC for Amiga"
For x% = 0 To 4
  Read sel$
  ~SendMessage(Dlg(1, 10), LB_ADDSTRING, 0, sel$)
  _Win$(DlgItem(1, 10)) = sel$
Next
ShowDialog # 1
Do
  Sleep
Until Me Is Nothing
Dlg 3D Off
```

```
Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  Switch Mess
  Case WM_COMMAND
    Switch LoWord(wParam)
    Case 10
      Trace "Notification code: " & HiWord(wParam)
    Case IDOK, IDCANCEL
      CloseDialog # 1
    EndSwitch
  EndSwitch
EndSub
```

Remarks

This command is particular useful for a dialog box in a GLL, because a GLL doesn't support OCX controls.

With the general **Control** statement any control type can be created.

See Also

Control, AnimateCtrl, AutoCheckBox, AutoRadioButton, CheckBox, ComboBox, CText, Dialog, DefPushButton, EditText, GroupBox, HeaderCtrl, ListBox, ListViewCtrl, LText, ProgressCtrl, PushButton, RadioButton, RichEditCtrl, RText, ScrollBar, StatusCtrl, TabCtrl, ToolBarCtrl, TrackBarCtrl, TreeViewCtrl, UpDownCtrl

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

TitleW Command

Purpose

Writes a string on the title line of a window.

Syntax

TitleW [#] n, txt$

wh:integer or Handle expression
txt$:sexp

Description

TitleW #n writes the string txt$ on the title line of the window *n*. *n* can have the values of 0 to _maxInt.

For a form created without a window number, **TitleW** takes its window handle from the **hWnd** property.

Example

```
TitleW # 1, " GFA-BASIC window "
OpenW # 1, 10, 10, 200, 100, -1
OpenW 2, 10, 120, 300, 100, -1
Win_2.Caption = " GFA-BASIC window "
```

Remarks

Windows opened with a number above 31 are accessed using the name **Form**(number). For instance, OpenW 40 is used as **Form**(40).**Activate** and the events have the format

Sub **Form_Activated**(Index%) where Index% the number of the form specifies.

See Also

SizeW, CloseW, MoveW, TopW, FullW, ClearW, OpenW

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

FullW Method

Purpose

Expands a Form or window to its maximum size

Syntax

FullW #n

Form.**FullW**

n:iexp

Description

Form.**FullW** expands a **Form** to full screen size.

FullW #n expands a window with given number (0 to _maxInt) to full screen size or opens such a window. When the window doesn't have number, the window handle can be specified as well.

Example

```
OpenW 1, 10, 10, 200, 100, -1 : Win_1.AutoRedraw =
  1
Local a%
FontSize = 10
Text 20, 20, "Please press a key"
KeyGet a%
Win_1.FullW  // or FullW #1
KeyGet a%
CloseW # 1
```

opens a small window, and after a key press maximizes it

See Also

Maximize, Minimize, OpenW, ChildW, ParentW

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

TopW Command

Purpose

Activates a window.

Syntax

TopW wh

wh:integer or Handle expression

Description

When several windows are opened, #n activates the window (0 to _maxInt) specified in n. If this window is covered by another, it is brought up to the front.

For a form created without a window number, **TopW** takes its window handle from the **hWnd** property.

The output is not redirected after a **TopW**. The output continues to go to the current active object (**Form** or **Printer**). The output can be redirected by Output =, Set Me =, or to a form using the **Activate** property.

TopW is synonym to the **Form** method **ToTop**.

A form brought to the top using **TopW** or **ToTop** will not be brought before a Form that has the **OnTop** property set.

Example

```
OpenW 1, 0, 0, 200, 100
Win_1.Caption = "Win/Form 1"
```

```
OpenW 2, 200, 0, 200, 100
Win_2.Caption = "Win/Form 2"
Win_2.AutoRedraw = 1
OpenW 3, 0, 100, 200, 100
Win_3.Caption = "Win/Form 3"
OpenW 40, 200, 100, 200, 100
Form(40).AutoRedraw = 1
Form(40).Caption = "Win/Form 40"
TopW # 2
Print "Result in Win 400"
Output = Win_2 ' or Win_2.Activate
Print "Result in Win 2"
Local a% : KeyGet a%
CloseW # 2
CloseW # 40
CloseW # 3
CloseW # 1
```

Remarks

Windows opened with a number above 31 are accessed using the name **Form**(number). For instance, OpenW 40 is used as **Form**(40).**Activate** and the events have the format Sub **Form_Activated**(Index%) where Index% specifies the number of the form.

See Also

SizeW, CloseW, MoveW, TitleW, FullW, ClearW, OpenW

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

AdjustW Command, Adjust Method

Purpose

Adjusts the window/Form size based on the given window client area.

Syntax

AdjustW wh, w, h

Form**.Adjust** w, h

wh, w ,h:integer expression

Description

A Windows window is composed of a number of elements (scroll bars, close box, minimize box, window client area, window rectangle, ...), which determine the window appearance and, as a rule, are used by Windows. An application program must simply configure the window with various attributes.

The point of reference for window manipulation, such as size change, is frequently the window client area. The GFA-BASIC command **AdjustW** and the Form method **Adjust** change the dimensions of the window. Two parameters are used for this purpose: the first specifies the width and the second the height of the client area in pixels.

Example

```
OpenW # 1, 10, 10, 400, 300, -1
Print "Press key to adjust Window"
Do : Sleep : Until Win_1 Is Nothing

Sub Win_1_KeyPress(Ascii&)
  AdjustW 1, 200, 150
  ' or Me.Adjust 200, 150
  ' or Win_1.Adjust 200, 150
  Cls : Print "Close Window"
EndSub
```

Changes the window size so that the work area becomes 200 pixels wide and 150 pixels high.

Remarks

AdjustW corresponds to the Windows function *AdjustWindowRect*().

See Also

Form, SizeW, MoveW, GetWinRect

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

ArrangeIcons Command

Purpose

Arranges iconized MDI Child windows (minimized Child windows) within a given window in rows.

Syntax

ArrangeIcons wh%

wh%:integer expression

Description

When a (Child-) window is iconized by clicking on the minimize box, the window icon frequently does not appear at the desired spot. Such icons can be moved by using the mouse. A convenient arrangement of icons consists of ordering them next to each other along the full window width.

When the window edge is reached the remaining icons are placed on the second, third, etc... row. The GFA-BASIC command **ArrangeIcons** performs such a placement and the only parameter it requires (wh%) is the number of the parent window.

Example

```
ParentW 1
ChildW 2, 1
ChildW 3, 1
Do
```

```
  Sleep                //move the iconized window
Until MouseK = 2
ShowW 2, SW_MINIMIZE
ShowW 3, SW_MINIMIZE
ArrangeIcons 1
Do : Sleep : Until Me Is Nothing
CloseW 1 : CloseW 2 : CloseW 3
```

Opens a parent window (1) and sets two child windows (2 and 3) within it. The child windows are minimized (iconized). The program then waits for a right mouse button click. Please move an icon in the parent window. Following the mouse click the command ArrangeIcons "rearranges" the icons.

Remarks

Instead of using this old command, you could use the Form method **Win_1**.**MdiArrangeIcons.** Other MDI Form methods are **MdiCascade** and **MdiTile**.

ArrangeIcons corresponds to the Windows function *ArrangeIconicWindows*().

See Also

ParentW, ChildW, Form

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Win Command

Purpose

Selects a window for output.

Syntax

Win n

n:iexp

Description

Win n selects window n for output, without activating it. The only parameter is a window number (0..31) as specified in **OpenW**, **ChildW**, and **ParentW**, or a window handle. Internally, GFA-BASIC 32 performs **Set Me = Form**(n)

For a form created using **Form** or **LoadForm**, you should pass the **hWnd** property.

Example

```
OpenW 1
Win_1.AutoRedraw = 1
Win_1.Caption = "#1"
OpenW 2 : TitleW 2, "#2"
Win 1    ' or Win Win_1.hWnd
Print "Hi"
Do
  Sleep
Until Win_1 Is Nothing Or Win_2 Is Nothing
CloseW 1 : CloseW 2
```

Remarks

Win is implemented for compatibility reasons and should be replaced with **Set Me =**, or **Output =**.

See Also

Win, Me, Form, Output

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

GetWinRect Command

Purpose

Returns the origin as well as the width and height of a window including its border.

Syntax

GetWinRect wh%, x%, y%, w%, h%

Description

The command **GetWinRect** returns in variables x% and y% the X and Y coordinates of the upper left window corner. w% contains the width and h% the height of the window in pixels. The window number (0 .. 31) is specified in wh%, any other value is considered a window handle.

Example

```
OpenW 1
Local h%, w%, x%, y%
GetWinRect 1, x%, y%, w%, h%
Print x%, y%, w%, h%
```

Remarks

GetWinRect corresponds in part to the GFA-BASIC command WINDGET 0,x%,y%,w%,h%. However, **WindGet** requires the output to be first redirected to a window by using **Win**(wh%).

See Also

WindGet

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

WindGet and WindSet Commands

Purpose

Returns and sets window parameters. Only for porting of GFA-BASIC 16 programs.

Syntax

WindGet i, a[,b[,c...]]

a = **Wind_Get(*i*)**

WindSet i, a[,b[,c...]]

i:integer expression;

a, b, c:variables;

Description

WindGet i, a, .. reads various window related parameters. The first parameter specifies the position to start reading. The number of variables that follow, determine the number window parameters to read. **Wind_Get**() only returns one parameter.

In addition to retrieving window information, **WindSet** can be used to set a number (marked with * in the list below).

>position of horizontal slider (0..1000)

i	Parameter
0	outer X-coordinate

1	outer Y-coordinate
2	outer width
3	outer height
4	inner X-coordinate
5	inner Y-coordinate
6	inner width
7	inner height
8*	position of vertical slider (0..1000)
9*	size of slider area
10*	
11*	size of slider area
12	reads the window attributes (as set with OpenW)
13*	reads the attributes of the pressed window button (from WINDSET)
14	character height (for example 8, 14, 16)
15	not available, was: character set address
16	not available, was: number of top window
17	not available, was: number of second to top window
18	not available, was: number of second to bottom window
19	not available, was: number of bottom window
20	text width

The asterisk indicates which parameters can be changed with **WindSet**.

Example

```
Local Int ha, hi, wa, wi, xa, xi, ya, yi
OpenW 1, 0, 0, 400, 300, -1
Win_1.AutoRedraw = 1
WindGet 0, xa, ya, wa, ha
WindGet 4, xi, yi, wi, hi
Print xa`ya`wa`ha
Print xi`yi`wi`hi
Print Wind_Get(8)``Wind_Get(9)
Print Wind_Get(10)``Wind_Get(11)
Print Hex(Wind_Get(12), 8)
Print Wind_Get(14)
Print Wind_Get(20)
Print
WindSet 9, 2000 : WindSet 11, 1500
Print Wind_Get(9), Wind_Get(11)
Do
  Sleep
Until Me Is Nothing
```

Remarks

The position and size of a window can be modified with **MoveW** and **SizeW.**

See Also

Form, WindSet

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

SendMessage Command

Purpose

Sends a message to one or more windows. The **SendMessage** command does not return before the message has been processed. After receiving a **SendMessage** call Windows immediately calls the relevant window function.

Syntax

SendMessage hWnd, Mess, wParam, lParam [,RetVal]

RetVal **= SendMessage(**hWnd, Mess, wParam, lParam)

hWnd, Mess, wParam, lParam, RetVal: integer expressions

Description

Windows uses messages to communicate between different parts of a program and/or different programs. They either contain the information about the current operation of the program or pass information from other applications. Such messages are managed by Windows in a buffer which is called the message queue.

The **SendMessage** command contains four parameters. The first parameter (hWnd) specifies the window handle to be included in the message or, if a GFA-BASIC window, the GFA-BASIC 32 window number. If hWnd contains HWND_BROADCAST the message is posted to all overlapping windows in the system.

The second parameter (Mess) contains the message (for example WM_SYSKEYUP). The two last parameters contain additional message information. The first one (wParam) is an unsigned 32 bit value, the second (lParam) an unsigned 32 bit value. The final parameter [RetVal] is an optional return value coming from the window that the send message was addressed to.

For example, in case of WM_SYSKEYUP wParam contains the virtual key code and lParam the scan code, repeat counter, the original keyboard state and other additional information (for example whether the key is a function key). The message sent with the **SendMessage** command is processed immediately and is not posted in the message queue.

SendMessage command is processed instantly and is not put in the message queue.

Example

```
// activate the Help mode, simulates a press
// on the Help [?]-button
OpenW 1
CenterMouse Win_1
SendMessage Me.hWnd, WM_SYSCOMMAND,
  SC_CONTEXTHELP, 0
Do
  Sleep
Until Me Is Nothing
```

Remarks

The GFA-BASIC command **SendMessage** corresponds to Windows function call *SendMessage*().

See Also

PostMessage

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

PostMessage Command

Purpose

Posts a message to the message queue of an application program (window). The function does not wait (like SendMessage) for the message to be processed. The posted message is retrieved by calling the Windows functions *GetMessage*() or *PeekMessage*().

Syntax

PostMessage hWnd, Msg, wParam, lParam

hWnd, Msg, wParam, lParam:integer expression

Description

Windows uses messages to communicate between different parts of a program and/or different programs. They either contain the information about the current operation of the program or pass information from other applications. Such messages are managed by Windows in a buffer which is called the message queue.

The **PostMessage** command contains four parameters. The first parameter (hWnd) specifies the window handle to be included in the message. If hWnd = HWND_BROADCAST ($FFFF) the message is posted to all overlapping windows in the system. The second parameter (Msg) contains the message (for example WM_SYSKEYUP). The two last 32-bit parameters wParam and lParam contain additional message information.

For example, in case of WM_SYSKEYUP wParam contains the virtual key code and lParam the scan code, repeat counter, the original keyboard state, and other additional information (for example whether the key is a function key). The message sent with the **PostMessage** command is posted in the message buffer (Queue) of the application (window) specified in hWnd and is not processed immediately (as is the case for **SendMessage**).

Example

```
OpenW # 1, 100, 100, 200, 200, ~15
DefFill 5
PRBox 0, 0, _X, _Y
Do
  Sleep
Until Me Is Nothing

Sub Win_1_KeyPress(Ascii&)
  PostMessage Me.hWnd, WM_CLOSE, 0, 0
EndSub
```

Remarks

The GFA-BASIC command PostMessage corresponds to built-in Windows function call *PostMessage*().

See Also

SendMessage

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

GetDevCaps Function

Purpose

Returns information about the current output device.

Syntax

% = **GetDevCaps**(n%)

n%:integer expression

Description

GetDevCaps returns information such as the screen resolution, number of colors etc. After the function call the information is returned in c%. The type of desired information is specified in n%, where n% can assume one of the following values:

n% = 0 or DRIVERVERSION - returns the driver version number

n% = 2 or TECHNOLOGY - returns the type of output device as follows:

c% = 1 or DT_RASDISPLAY	Raster display (screen)
c% = 2 or DT_RASPRINTER	Raster printer (dot matrix printer)
c% = 3 or DT_RASCAMERA	Raster camera (screen photo or similar)
c% = 4 or DT_CHARSTREAM	Character stream (e.g. file)

c% = 5 or DT_METAFILE	Metafile (.WMF)
c% = 6 or DT_DISPFILE	Display file (cards ??)

n% = 4 or HORZSIZE - Width of physical display in mm

n% = 6 or VERTSIZE - Height of physical display in mm

n% = 8 or HORZRES - Width of physical display in pixels

n% = 10 or VERTRES - Height of physical display in pixels

n% = 88 or LOGPIXELSX - Number of pixels per logical inch on the x-axis (horizontal DPI resolution)

n% = 90 or LOGPIXELSY - Number of pixels per logical inch on the y-axis (vertical DPI resolution)

n% = 12 or BITSPIXEL - Number of bits per pixel

n% = 14 or PLANES - Number or color planes

n% = 16 or NUMBRUSHES - Number of device specific fill patterns (Brushes)

n% = 18 or NUMPENS - Number of device specific line styles (Pens)

n% = 22 or NUMFONTS - Number of device specific fonts

n% = 24 or NUMCOLORS - Number of available colors

n% = 40 or ASPECTX - relative width of a pixel for lines

n% = 42 or ASPECTY - relative height of a pixel for lines

n% = 44 or ASPECTXY - diagonal size of a pixel for lines n% = 26 or PDEVICESIZE - size of internal data structure (PDEVICE)

n% = 36 or CLIPCAPS - indicates if the output device can perform rectangle clipping. c% = 1 for yes or 0 for no

n% = 104 or SIZEPALETTE - Number of colors in the system palette

n% = 106 or NUMRESERVED - Number of reserved entries in the system palette

n% = 108 or COLORRES - Color resolution in bits per pixel

n% = 38 or RASTERCAPS - A word of information about the capabilities of the output device to display raster graphics. The following values are possible:

c% = 2 or RC_BANDING	requires banding (printer - Escape NEWBAND)
c% = 4 or RC_SCALING	allows scaling
c% = 8 or RC_BITMAP64	allows bitmaps of more than 64 KBytes
c% = $0010 or RC_GDI20_OUTPUT	allows Windows 2.0 functions
c% = $0080 or RC_DI_BITMAP	allows GetDIBits and SetDIBits
c% = $0100 or RC_PALETTE	contains a palette
c% = $0200 or RC_DIBTODEV	allows DIBitsToDevice
c% = $0800 or RC_STRETCHBLT	allows StretchBlt
c% = $2000 or	allows StretchDIBits

RC_STRETCHDIB

n% = 28 or CURVECAPS - A word of information about the capabilities of output devices to display curves. A set bit indicates the following:

0 - Circle
1 - Arc
2 - Arc
3 - Ellipse
4 - Wide lines
5 - Line style
6 - Wide line
style

n% = 30 or LINECAPS - A word of information about the capabilities of output devices to display line styles. A set bit indicates the following:

1 - Polylines
4 - Wide Lines
5 - Line styles
6 - Wide line styles
7 - Solid area (Plotters can't do
this)

Bit 0, 2 and 3 are reserved.

n% = 32 or POLYGONALCAPS - A word of information about the capabilities of output devices to display polygons. A set bit indicates the following:

0 - Alternating fill
mode
1 - Rectangle
2 - Winding fill mode
3 - Scan line

4 - Wide line
5 - Line style
6 - Wide line style
7 - Internal area

n% = 34 or TEXTCAPS - A word of information about the capabilities of output devices to display text. A set bit indicates the following:

0 - Character positioning
1 - Stroke positioning (exact)
2 - Stroke clipping (exact)
3 - Character rotation in 90° steps
4 - Arbitrary character rotation
5 - Separate x and y scaling
6 - Double characters for scaling
7 - Integer scaling
8 - Arbitrary scaling
9 - Double width characters
10 - Italics
11 - Underline
12 - Strike out
13 - Bitmap fonts (Raster fonts)
14 - Vector fonts
15 - reserved 0

Example

```
OpenW # 1
Print GetDevCaps(HORZRES)
Print GetDevCaps(VERTRES)
```

prints the horizontal and vertical screen resolution in pixels.

Remarks

GetDevCaps() corresponds to the Windows API function *GetDeviceCaps*().

Most of the information can be obtained using the **Screen** object or **App** object.

See Also

Screen, App

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

SysMetric Function

Purpose

Returns the dimensions of a specified element.

Syntax

% = **SysMetric**(e%)

e%:integer expression

Description

The **SysMetric**() function returns the dimensions of the element specified in e% in pixels. e% must take one of the following values:

SM_CXSCREEN (0) - screen width

SM_CYSCREEN (1) - screen height

SM_CXFRAME (32) - window rectangle width, for "sizing"

SM_CYFRAME (33) - window rectangle height, for "sizing"

SM_CXVSCROLL (2) - width of arrow fields in vertical scrollbars

SM_CYVSCROLL (20) - height of arrow fields in vertical scroll bars

SM_CXHSCROLL (21) - width of arrow fields in horizontal scroll bars

SM_CYHSCROLL (3) - height of arrow fields in horizontal scroll bars

SM_CYCAPTION (4) - title list height

SM_CXBORDER (5) - window rectangle width

SM_CYBORDER (6) - window rectangle height

SM_CXDLGFRAME (7) - width of a Dialog box frame

SM_CYDLGFRAME (8) - height of a Dialog box frame

SM_CXHTHUMB (10) - width of the vertical scroll bar

SM_CYVTHUMB (9) - height of the vertical scroll bar

SM_CXICON (11) - icon width

SM_CYICON (12) - icon height

SM_CXCURSOR (13) - cursor width

SM_CYCURSOR (14) - cursor height

SM_CYMENU (15) - height of a single line menu bar

SM_CXFULLSCREEN (16) - maximum width of the window client area

SM_CYFULLSCREEN (17) - maximum height of the window client area

SM_CYKANJIWINDOW (18) - height of a Kanji window

SM_CXMINTRACK (34) - minimum tracking width of a window

SM_CYMINTRACK (35) - minimum tracking height of a window

SM_CXMIN (28) - minimum width of a window

SM_CYMIN (29) - minimum height of a window

SM_CXSIZE (30) - width of bitmap in the title bar

SM_CYSIZE (31) - height of bitmap in the title bar

SM_MOUSEPRESENT (19) - not zero if mouse is installed.

SM_MOUSEWHEELPRESENT (75) - not zero if mouse wheel is present.

SM_DEBUG (22) - not zero in case of Windows debugging version.

SM_SWAPBUTTON (23) - not zero if the left and right mouse buttons are swapped.

SM_CXDOUBLECLK (36) - width in pixels, for double clicking sequence.

SM_CYDOUBLECLK (37) - height in pixels, for double clicking sequence.

SM_CXICONSPACING (38) - distance width between the icons, in pixels

SM_CYICONSPACING (39) - distance height between the icons, in pixels

SM_MENUDROPALIGNMENT (40) - true if right aligned menus are in use

SM_PENWINDOWS (41) - true, if a Pen Windows is in use

SM_DBCSENABLED (42) - true, if a double byte character is in use (far east - actually GB32 doesn’t work correct in this direction).

SM_CMOUSEBUTTONS (43) - number of mouse buttons (2, 3, or 0).

SM_CXDLGFRAME The width of a frame of a non-sizeable window with title.

SM_CXSIZEFRAME (32) - SM_CXFRAME window rectangle width, for "sizing".

SM_CYSIZEFRAME (33) - SM_CYFRAME window rectangle height, for "sizing".

SM_SECURE (44) - true is the security package is available.

SM_CXEDGE (45) - the dimension of a 3D border in pixels

SM_CYEDGE (46) - the dimension of a 3D border in pixels

SM_CXMINSPACING (47) - width of the distance between symbols (NT 3.51)

SM_CYMINSPACING (48) - height of the distance between symbols (NT 3.51)

SM_CXSMICON (49) - the width of a small icon

SM_CYSMICON (50) - the height of a small icon

SM_CYSMCAPTION (51) - the height of a small title in pixels.

SM_CXSMSIZE (52) - the width of the buttons in the title of windows, given in pixels.

SM_CYSMSIZE (53) - the height of the buttons in the title of windows, given in pixels.

SM_CXMENUSIZE (54) - the width of the buttons, etc. in the title of a MDI child window, in pixels

SM_CYMENUSIZE (55) - the height of the buttons, etc. in the title of a MDI child window, in pixels

SM_ARRANGE (56) - the minimizing position of windows
0 starts left hand below (Default Position).
1 starts right hand below
2 starts top left
3 starts top right
0+4 starts left below and forward in vertical direction to the top
1+4 starts right below and forward in vertical direction to the top
2+4 starts top left, vertical
3+4 starts top right, vertical
8 during minimizing the window is hidden

SM_CXMINIMIZED (57) - width of a minimized window in pixels.

SM_CYMINIMIZED (58) - height of a minimized window in pixels.

SM_CXMAXTRACK (59) - maximum width of a window during changing it’s size (with the mouse).

SM_CYMAXTRACK (60) - maximum height of a window during changing it’s size (with the mouse).

SM_CXMAZIMIZED (61) - width of a maximized window in pixels.

SM_CYMAXIMIZED (62) - height of a maximized window in pixels.

SM_NETWORK (63) - true, if a Windows network is active. An odd value for a known on, otherwise an even one.

SM_CLEANBOOT (67) - 0 = normal,1 = protected mode, 2 = protected with network.

SM_CXDRAG (68) - the width of a range which will be used to allow a minimum movement of the mouse while double clicking, without a drag (to start the movement with the mouse) is happend.

SM_CYDRAG (69) - the height of a range which will be used to allow a minimum movement of the mouse while double clicking, without a drag (to start the movement with the mouse) is happend.

SM_SHOWSOUNDS (70) - true, if Windows use optical messages instead of acoustic ones; for people with a hearing damage and circumstances permitting for the usage in an open plain office (can be controlled from the system control).

SM_CXMENUCHECK (71) - width of the hook let in pixels for corresponding menu entries.

SM_CYMENUCHECK (72) - height of the hook let in pixels for corresponding menu entries.

SM_SLOWMACHINE (73) - true, it the computer is slow (subjective scoring of the operating system).

SM_MIDEASTENABLED (74) - true in the far east.

SM_CMETRICS (76) - value of numbers for the parameter GetSystemMetrics.

Example

```
Debug.Show
Trace SysMetric(SM_CXCURSOR)
```

Returns the width of the cursor

Remarks

The **Screen** Object returns the same values and more.

See Also

Screen Object, App Object

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

MousePointer Property

Purpose

Returns or sets a value indicating the type of mouse pointer displayed when the mouse is over a particular part of an object at run-time.

Syntax

Object.**MousePointer** [= value]

Object:Ocx Object
value:iexp

Description

The settings for *value*, which are NOT declared by GFA, are:

Constant	**Value**	**Description**
basDefault	0	(Default) Shape determined by the object.
basArrow	1	Arrow.
basCross	2	Cross (crosshair pointer).
basIbeam	3	I beam.
basIcon	4	Icon (small square within a square).
basSize	5	Size (four-pointed arrow pointing north, south, east, and west).
basSizeNESW	6	Size NE SW (double arrow pointing northeast and southwest).

basSizeNS	7	Size N S (double arrow pointing north and south).
basSizeNWSE	8	Size NW SE (double arrow pointing northwest and southeast).
basSizeWE	9	Size W E (double arrow pointing west and east).
basUpArrow	10	Up Arrow.
basHourglass or **basWait**	11	Hourglass (wait).
basNoDrop	12	No Drop.
basArrowHourglass	13	Arrow and hourglass.
basArrowQuestion or **basHelp**	14	Arrow and question mark.
basSizeAll	15	Size all.
basCursor	98	Custom icon specified by the MouseCursor (LoadCursor) property.
basCustom	99	Custom icon specified by the **MouseIcon** (Picture) property.

Example

```
OpenW 1
MousePointer = 11 // basHourClass
Do
  Sleep
Until Me Is Nothing
```

Remarks

See Also

MouseCursor, MouseIcon, MousePointer, DefMouse

{Created by Sjouke Hamstra; Last updated: 24/01/2019 by James Gaite}

DefMouse Command

Purpose

sets the mouse shape and appearance.

Syntax

DefMouse a%

a%:integer expression

Description

The possible values are synonymous to the mouse pointer objects, as follows:

0	Default	whatever is active will be used
1	Arrow	normal arrow
2	Cross	a little cross
3	IBeam	text cursor (looks like a big i with head line and under score)
4	Icon	don’t use, compatible Windows 3.1 icon
5	Size	general sizing exp. with the system menu (for windows)
6	SizeNESW	general sizing; double arrow from left down to upper right(cursor pointing northeast and southwest)
7	Size NS	general sizing; double arrow below to top (cursor pointing north and south)
8	Size NWSE	general sizing; double arrow from right down to upper left (cursor pointing

		northwest and southeast)
9	Size WE	general sizing, double arrow from right hand to left hand (cursor pointing west and east)
10	Up-Arrow	arrow up (cursor)
11	Hourglass	hourglass cursor
12	No Drop	prohibition sign (like under Windows explorer and copying: not allowed to place something there)
13	Arrow Hourglass	arrow with hourglass (cursor)
14	Help	arrow with question mark (help)
15	Size All	size all (cursor), four fold arrow
98	Cursor	Use the mouse set with the Form.**MouseCursor** property
99	Custom	Use the mouse set with the Form.**MouseIcon** property

DefMouse 0 is the default.

It makes more sense to use the Form properties .**MouseIcon** or .**MousePointer** to set the mouse pointer for a form, window, or dialog. Rather than DefMouse 11, you would use Form.**MousePointer** = 11.

Example

```
OpenW 1
Local mk%, mx%, my%
Do
  Mouse mx%, my%, mk%
  Exit If mk% = 2
  /* quarter top left
  If mx% <= _X / 2
    If my% <= _Y / 2
```

```
      DefMouse 2
      /* mouse symbol = Cross
    Else
      /* quarter left down
      DefMouse 3
      /* mouse symbol = IBeam
    EndIf
  Else
    /* quarter upper right
    If my% <= _Y / 2
      DefMouse 11
      /* mouse symbol = hourglass
    Else
      /* quarter right down
      DefMouse 10
      /* mouse symbol = arrow up
    EndIf
  EndIf
Loop
CloseW 1
```

See Also

Mouse, MouseX, MouseY, MouseK, HideM, ShowM

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

HideM, ShowM Commands

Purpose

Hides and unhides (shows) the mouse pointer.

Syntax

HideM

ShowM

Example

```
OpenW 1
HideM
Delay 2
ShowM
Delay 1
CloseW 1
```

16 bit example:

```
OpenW # 1
Print "To end press a mouse key"
Local border%, hidden?, mk%, mx%, my%
border% = _Y >> 1
hidden? = False
Do
  Sleep
  Mouse mx%, my%, mk%
  If my% > border%
    HideM
    hidden? = True
  Else If (my% <= border%) %& (hidden?)
```

```
    ShowM
    hidden? = False
  EndIf
Loop Until mk%
If hidden? Then ShowM
CloseW # 1
```

Hides the mouse pointer as long as it moves within in the window, outside it’s still visible.

See Also

DefMouse, Mouse, MouseX, MouseK

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

SetCapture and ReleaseCapture Command

Purpose

Sets and releases exclusive mouse input which can be limited to a specific window.

Syntax

ReleaseCapture

SetCapture wh%

wh%:integer expression

Description

SetCapture redirects all mouse events - regardless of how many windows are currently open - to the window with the number wh% (0..31) or API handle. For a **Form** object use the **hWnd** property.

SetCapture can only be invoked for one single window at any one time. If the mouse input is to be redirected to another window, the command **ReleaseCapture** must first be called for the current (mouse) window and then **SetCapture** for the new window.

ReleaseCapture redirects the mouse input back to the corresponding window (the window with the mouse pointer).

Example

```
OpenW 1 : Win_1.Caption = Win_1.Name
OpenW 2 : Win_2.Caption = Win_2.Name
SetCapture 1// or Win_1.hWnd
Do
  Sleep
  If MouseK %& 1
    Print MouseX, MouseY
  EndIf
Until MouseK %& 2        'press right mouse button
ReleaseCapture
CloseW 2
CloseW 1
```

Causes all mouse-input to be sent only to window 1 until right mouse button is pressed

Remarks

SetCapture and **ReleaseCapture** correspond to Windows function SetCapture() and ReleaseCapture respectively.

See Also

-

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Mouse Command

Purpose

Returns the current X and Y coordinates of the mouse pointer, the status of the mouse buttons and (optionally) the status of the keyboard shift keys (KB shift).

Syntax

Mouse mx%, my%, mk% [, kb%]

mx%, my%, mk%, kb%:ivar

Description

The **Mouse** command is a combination of GFA-BASIC functions **MouseX**, **MouseY**, **MouseK**, and **MouseKB**. In addition, by supplying the fourth optional parameter the status of the keyboard shift keys can also be interrogated.

Example

```
OpenW 1
Local a$, ak%, ax%, ay%, mk%
Local b%, mx%, my%, t$, title$, ab%
Do
  t$ = InKey$
  // to end press ESC
  Exit Do If Asc(t$) = 27
  title$ = "X-direction" + Space(10)
  title$ = title$ + "Y-direction"
  title$ = title$ + Space(10)
  title$ = title$ + "Mouse Key" + Space(10)
  title$ = title$ + "Shift-Key"
  Text 8, 16, title$
  ax% = mx%, ay% = my%, ak% = mk%, ab% = b%
  Mouse mx%, my%, mk%, b%
  // only new, if something will change
  If (ax% <> mx%) || (ay% <> my%) _
    || (ak% <> mk%) || (ab% <> b%)
    // delete a line
    Text 8, 46, Space$(999)
    a$ = Str$(mx%, 4, 0) + Space(25)
    a$ = a$ + Str$(my%, 4, 0) + Space(20)
    a$ = a$ + Str$(mk%, 8, 0) + Space(20)
    a$ = a$ + Str$(b%)
    Text 8, 46, a$
  EndIf
Loop
CloseW 1
```

See Also

DefMouse, MouseK, MouseKB, MouseX

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

MouseX, MouseY Function

Purpose

Returns the current X coordinate of the mouse pointer.

Syntax

mx% = **MouseX**

my% = **MouseY**

mx%, my%:ivar

Description

Returns the mouse position relative to the client area of the window.

Ocx controls that are aligned at the border of the parent form (using **Align**) change the **Scale** settings of the parent. **ScaleLeft** and **ScaleTop** are set to the top-left pixel of the uncovered client area of the form. **ScaleWidth** and **ScaleHeight** are set to width and height of the uncovered area. The mouse coordinates returned from **MouseX**, **MouseY** and that are passed in the forms **MouseMove**, **MouseUp**, and **MouseDown** events are relative to the new origin.

Example

```
OpenW 1, 100, 100, 200, 200, 0
Local ax%, ay%, mx%, my%
Do
```

```
  Sleep
  ax% = mx%, ay% = my%
  mx% = MouseSX, my% = MouseSY
  If (ax% <> mx%) || (ay% <> my%)
    Text 10, 10, "Mouse moving on the screen in"
    Text 10, 30, "X-direction" + Space(20) + "Y-
      direction"
    Text 10, 50, Space$(999)
    Text 20, 50, Str$(mx%) + Space(30) + Str$(my%)
    If mx% - 2 < TwipsToPixelX(Win_1.Left) Or mx% +
      2 > TwipsToPixelX(Win_1.Left + Win_1.Width) Or
      _
      my% - 2 < TwipsToPixelY(Win_1.Top) Or my% + 2
        > TwipsToPixelY(Win_1.Top + Win_1.Height)
      Text 10, 90, Space$(999)
      Text 10, 110, Space$(999)
      Text 10, 130, Space$(999)
    EndIf
  EndIf
Loop Until MouseK = 2
CloseW 1

Sub Win_1_MouseMove(Button&, Shift&, x!, y!)
  Text 10, 90, "Mouse moving in the window at"
  Text 10, 110, "X-direction" + Space(20) + "Y-
    direction"
  Text 10, 130, Space$(999)
  Text 20, 130, Str$(x!) + Space(30) + Str$(y!)
EndSub
```

Remarks

A real application would use the _MouseMove event sub.

See Also

DefMouse, Mouse, MouseSX, MouseK

{Created by Sjouke Hamstra; Last updated: 19/10/2014 by James Gaite}

MouseK (Property), MouseKB Function

Purpose

Returns the current status of the mouse buttons and shift keys.

Syntax

mk% = [*Screen*.]**MouseK**

ms% = **MouseKB**

mk%, ms%:ivar

Description

The **MouseK** function returns the status of the mouse buttons.

1 - Left button pressed

2 - Right button pressed

3 - Both pressed

The **MouseKB** function returns the current status of the shift keys of your keyboard.

1 - Left and/or Right Shift

2 - Left and/or Right Ctrl

4 - Left Alt key

6 - Right Alt key

Example

```
OpenW 1
Local a$
a$ = "please press Shift key + mousekey"
TitleW 1, a$
PrintScroll = True
Do
  Do
  Loop Until MouseK
  Print MouseK, MouseKB
  Print "to end with mouse + Escape-key"
Loop Until GetAsyncKeyState(27) < 0
CloseW 1
```

Remarks

MouseKB uses the *GetAsyncKeyState*() API function.

Known Issues

Pressing the right Alt key is the same as pressing Ctrl and the left Alt key so the two key combinations can not be told apart using **MouseKB**. To get a more accurate result use the **Screen.ShiftKeys** value.

Pressing the right Alt key, on occasions either a 2 or 4 will be returned by **MouseKB** rather than or as well as the expected 6. This can seen by running the example below and contunally pressing the right Alt key.

```
Local Int mk, ms, oms
```

```
Debug.Show
Do
  mk = MouseK
  ms = MouseKB
  If ms = 0 : oms = 0
  ElseIf ms <> 0 And ms <> oms : Debug ms;" - "; :
    oms = ms
    If ms >= 6 Then Debug.Print "Right Alt key"; :
      Xor ms, 6 : If ms <> 0 Then Debug.Print " & ";
    If Btst(ms, 0) Then Debug.Print "Shift key"; :
      If ms <> 1 Then Debug.Print " & "; : Xor ms, 1
    If Btst(ms, 1) Then Debug.Print "Ctrl key"; :
      If ms <> 2 Then Debug.Print " & "; : Xor ms, 2
    If Btst(ms, 2) Then Debug.Print "Left Alt key";
    Debug.Print
  EndIf
Loop Until mk <> 0
Debug.Hide
```

To get around this problem, use the **Screen.ShiftKeys** value.

See Also

Screen, Mouse, MouseX, MouseSY

{Created by Sjouke Hamstra; Last updated: 19/10/2014 by James Gaite}

MouseSX, MouseSY Functions, MouseX, MouseY Properties

Purpose

Returns the current x and y position of the mouse in pixels relative to the upper left corner of the current form and/or Desktop.

Syntax

x% = **MouseSX**
y% = **MouseSY**

x% = Screen.**MouseX**
y% = Screen.**MouseY**

x%, y%:ivar

Description

MouseSX and **MouseSY** are shortcuts for the **Screen** properties **MouseX**, **MouseY** and return the mouse coordinates within the desktop.

Example

```
OpenW 1, 100, 100, 200, 200, 0
Local ax%, ay%, mx%, my%
Do
  Sleep
  ax% = mx%, ay% = my%
  mx% = MouseSX, my% = MouseSY
  If (ax% <> mx%) || (ay% <> my%)
    Text 10, 10, "Mouse moving on the screen in"
    Text 10, 30, "X-direction" + Space(20) + "Y-
      direction"
    Text 10, 50, Space$(999)
    Text 20, 50, Str$(mx%) + Space(30) + Str$(my%)
    If mx% - 2 < TwipsToPixelX(Win_1.Left) Or mx% +
      2 > TwipsToPixelX(Win_1.Left + Win_1.Width) Or
      _
      my% - 2 < TwipsToPixelY(Win_1.Top) Or my% + 2
        > TwipsToPixelY(Win_1.Top + Win_1.Height)
      Text 10, 90, Space$(999)
      Text 10, 110, Space$(999)
      Text 10, 130, Space$(999)
    EndIf
  EndIf
Loop Until MouseK = 2
CloseW 1

Sub Win_1_MouseMove(Button&, Shift&, x!, y!)
  Text 10, 90, "Mouse moving in the window at"
  Text 10, 110, "X-direction" + Space(20) + "Y-
    direction"
  Text 10, 130, Space$(999)
  Text 20, 130, Str$(x!) + Space(30) + Str$(y!)
EndSub
```

Remarks

The **MouseSX** and **MouseSY** internally call the Windows function *GetCursorPos*() and therefore require no **PeekEvent**, **GetEvent** or **Sleep**.

See Also

MouseX, Mouse, MouseK, Screen

{Created by Sjouke Hamstra; Last updated: 19/10/2014 by James Gaite}

Keyget Command

Purpose

Gets the first character in the keyboard buffer.

Syntax

Keyget n

Description

Keyget n% waits for a key press and returns in n% a long word with the following layout:

Bit 0 to 7 - ASCII code

Bit 8 to 15 - Scan code

For a list of Scan and ASCII codes, see Key Codes and ASCII Values.

Example

```
Local n%
OpenW # 1
Print "Press any key"
KeyGet n%
Print Hex$(n%, 4)
```

Waits for key entry and then returns the codes of the pressed key.

Remarks

During the waiting period for a key to hit, GFA-BASIC 32 doesn't block other programs, but performs a **DoEvents**.

See Also

Inkey$, KeyTest

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

InKey$ Function

Purpose

Reads a character from the keyboard (excluding special keys like Shift, Alt, Alt Gr, Ctrl...).

Syntax

string = **InKey[**$]

Description

InKey$ does not wait for a key press but, if no keys were pressed since the last keyboard request (by the processor), it returns an empty string. Otherwise, **InKey**$ reports the ASCII code of the pressed key.

If the pressed key has no ASCII code (the special keys, for example), the scan code of the pressed key is returned instead. If this is this case a two character string is returned, the first of which is a **Chr**$(0) and the second the corresponding key code.

For a list of Scan and ASCII codes, see Key Codes and ASCII Values.

Example

```
Local t$
OpenW # 1
Do
  t$ = InKey$
  Exit If Cvi(t$) = 6912 // Press Esc to quit
  If t$ <> ""
    If Len(t$) = 1                  //normal key
      Print "Key: "; t$; Spc(3);
      Print "ASCII code : "; Asc(t$)
    Else
      Print "Chr$(0), Scan code : "; Cvi(t$)
    EndIf
  EndIf
Loop
CloseW # 1
```

This example displays the ASCII or scan code for each pressed key.

Remarks

Instead of **InKey** you should use the Ocx event subs:

Sub Form_**KeyDown**(Code&, Shift&)

Sub Form_**KeyPress**(Ascii&)

Sub Form_**KeyUp**(Code&, Shift&)

Sub **Screen_KeyPreview**(hWnd%, uMsg%, wParam%, lParam%, Cancel?)

See Also

Keyget, KeyTest

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

KeyTest Command

Purpose

Reads the first character in the keyboard buffer.

Syntax

KeyTest n

Description

KeyTest n is similar to **InKey**$, that is to say, it reads a character from the keyboard when a key - other than Alt, Ctrl, Shift or Caps Lock - is pressed. If no key was pressed a 0 is returned. Otherwise the ASCII and scan code of the character is returned

Bit 0 to 7 - ASCII-Code

Bit 8 to 15 - Scan-Code.

For a list of Scan and ASCII codes, see Key Codes and ASCII Values.

KeyTest returns immediately; it doesn't wait for a keyboard hit. You should never use **KeyTest** in a serious program. **KeyTest** was built in to offer an easy function to use in simple test programs to get a character of the keyboard.

Example

```
OpenW 1
```

```
Local n%
Repeat
  KeyTest n%
  If Byte(n%)
    Print Byte(n%),
    Print LoByte(n%),
    Print HiByte(n%)
  EndIf
Until Byte(n%) = 27
CloseW # 1
```

The program loops until the ESCAPE key is pressed.

Remarks

See Also

Inkey$, Keyget

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

Alert Function

Purpose

Draws a message box on the screen.

Syntax

Alert *IconAndFlag, MainText$, DefButton, ButtonText$ [,RetVal]*

RetVal **= Alert**(*IconAndFlag, MainText$, DefButton, ButtonText$*)

IconAndFlag, DefButton	*:*	*iexp*
RetVal	*:*	*ivar*
MainText$, ButtonText$	*:*	*sexp*

Description

An **Alert** box is a special form of a message box. It is used when a point in a program is reached where the program is to be cancelled, a certain branch is to be taken, or some other user decision is to be made.

The first integer expression, *IconAndFlag*, determines which symbol will be included in the **Alert** box together with the message. The following symbols are available:

IconAndFlag	**Meaning**
0	mark symbol
1	stop mark
2	question mark

3	exclamation mark
4	information mark
5	windows flag
6	application mark
7	information mark
16	buttons are placed at the right border
32	shadow
64	text is right aligned
128	text is centered

MainText$ contains the message which is to be displayed in the Alert Box. If the text is too long for one line it can be split in up to 4 lines by using "|".

ButtonText$ contains up to five possible alternatives for user response.

DefButton indicates which of these alternatives the default is. This alternative is then selected by pressing the Return key. The alternatives are numbered from 1 to 5 and are separated from each other by a "|".

RetVal contains the number of the alternative which was actually selected.

Example

```
Auto a$, b$, i%, j%, retval%
OpenW # 1
i% = 2
a$ = "Which procedure should|be executed next"
j% = 1
b$ = "Input | Calculate | Print | File output |
  CANCEL"
retval% = 0
```

```
Alert 2 | 16, a$, j%, b$, retval%
CloseW # 1
```

Creates an Alert Box with a question mark as symbol and the message: "Which procedure should be executed next". The default alternative is "Input". The alternatives are:

Input, Calculate, Print, File output, and CANCEL.

retval% contains the number of the selected alternative.

Remarks

AlertBox is a synonym to **Alert** and can be used instead.

In addition to the menu bar and pop-up menus, the **Alert[Box]** is a third possible way of communication between the program and the user. Furthermore, it can prove useful when incorporated inside LG32 libraries as a customised messagebox, where OCX objects and Dialogs can not be used.

Known Issues

- In Windows 8, 8.1 and 10, the static text box (which holds MainText) and the icon image holder are drawn with white backgrounds; a patch has been created to solve this problem by Sjouke Hamstra and will be released in the near future.
  [Reported by James Gaite, 09/03/2017]

- Alert box does not recognise of multiple monitors and is always displayed on the main monitor. Use Prompt, InputBox or MsgBox instead or, in a GLL, use MsgBox0.
  [Reported by Sjouke Hamstra, 03/04/2018]

See Also

Menu, Popup, Message, MsgBox, Prompt

{Created by Sjouke Hamstra; Last updated: 04/04/2018 by James Gaite}

_Atom$ Function

Purpose

Returns the global name of a given atom.

Syntax

$ = **_Atom$**(id)

Description

Returns the String associated with an atom (a handle) in windows global atom table. This Function is just a wrapper around the Windows API function *GlobalGetAtomName*().

See Also

Atom Add, Atom Find, Atom Delete

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Atom Add Command

Purpose

Same as API function x%=*GlobalAddAtom*("name")

Syntax

Atom Add string, atom%

Description

The **Atom Add** function adds a character string to the global atom table and returns a unique value (an atom) identifying the string.

Example

```
Local atomApp%
Atom Add App.Name, atomApp%
Print atomApp%
```

See Also

Atom Delete, Atom Find, Atom$

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Atom Find Command

Purpose

Same as API function x%=*GlobalFindAtom*("name")

Syntax

Atom Find "name", atom%

Description

The **Atom Find** function searches the global atom table for the specified character string and retrieves the atom associated with that string.

Example

```
Local atomApp%
Atom Add App.Name, atomApp%
// ...other commands...
Atom Find App.Name, atomApp%
Print atomApp%
```

See Also

Atom Add, Atom Delete, Atom$

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Atom Delete Command

Purpose

Same as API function *GlobalDeleteAtom*("name")

Syntax

Atom Delete x%

Description

The **Atom Delete** function decrements the reference count of a global string atom. If the atom's reference count reaches zero, *GlobalDeleteAtom* removes the string associated with the atom from the global atom table.

Example

```
Local atomApp%
Atom Add App.Name, atomApp%
Atom Delete atomApp%
```

See Also

Atom Find, Atom Add, Atom$

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

reStop Command

Purpose

Ends a thread that is performing a regular expression operation (used in a **Try**/**Catch** structure)

Syntax

reStop

Description

reStop is used within a thread auxiliary to the main program to stop that thread when it is performing a regular expression operation. **reStop** itself must always be used in a Try/Catch guarded code block.

NOTE: The **reStop** command is just one method of ending a thread and should be used with caution if at all. See Creating and Terminating Threads for more information on threads and how to close them.

Example

```
Local i%, j%, t#, m$(100000)
Global ti%, End?
// Create a random string array
For i% = 1 To 100000 : m$(i%) = RandomString :
 Next i%
Local h As Hash Int
// Create the Thread
Debug.Print "Thread Handle =";CreateThread(0, 0,
 ProcAddr(Thread), 0, 0, V:ti%) & "    "
```

```
Debug.Print "Thread ID =";ti%
// Perform the match
Try
  t# = Timer
  j% = reMatch(m$(), "[a-z]+n", 0, i% - 1, h[])
Catch
  Debug.Print Err.Description
  j% = -1
EndCatch
// Close the thread and print results
t# = Timer - t#
End? = True
While ti% : Sleep 30 : Wend : Sleep 0
Debug.Print h[%]; " Matches in"; t#; "s"
If j% < 0 Then Print "Aborted"
For Each j% In h[]
  Debug.Print m$(j%)
Next
Debug.Show
Message "Results are on the Debug Screen"

Function RandomString
  Local n As Int32, txt$
  For n = 1 To Int(Rnd * 10) + 4 : txt$ = txt$ &
    Chr(Int(Rnd * 93) + 31) : Next n
  Return txt$
EndFunction

Sub Thread(p%)
  Debug.Print "Thread Started"
  Do
    If(GetAsyncKeyState(VK_ESCAPE) < 0)
      reStop
      Exit
    EndIf
    Sleep 100
  Loop Until End?
  ti% = 0
EndSub
```

The example program creates a random string array then searches the text for any succession of letters followed by the letter n.

Just before reMatch is executed - it should be placed in a Try/Catch/EndCatch construction - a second thread is created. The Debug.Print statement is unimportant; the only important thing is that the return value of the *CreateThread()* function non-zero.

When the regular expression search has ended, the global variable End? is set to True. This variable is inspected in the second thread and will cause the thread to end.

Meanwhile, inside the loop the thread tests for a key press (Esc key) and when it is pressed invokes the **reStop** command. **reStop** brings up an error message box (Err.Description = "reStop", Err.Number = 140). The error is trapped in the Try/Catch block in the main part of the program.

Remarks

NOTE: Breaking (pressing Ctrl-Break) a program which is using more than one thread or process will most likely result in an error in the IDE which will see GB32 closed down by Windows. Therefore, if you are testing such a program, always save it first.

See Also

Multithreading, preMatch, reMatch, reSub, Hash

{Created by Sjouke Hamstra; Last updated: 14/07/2015 by James Gaite}

Replace Funcrion

Requires: gfawinx.lg32

Purpose

Returns a string in which a specified substring has been replaced with another substring a specified number of times.

Syntax

str = **Replace**(src, find, replaceby [, start] [, count] [, compare])

str, find, replaceby	*: string expression*
src	*: string variable*
start, count, compare	*: integer expression*

Description

The *src* argument should be a string variable containing the string to be found that you wish to replace, *find* is the substring being searched for and *replaceby* the string that is inserted in its place. The optional *start* argument specifies the position within src where the search is to begin; if it is omitted, 1 or the first character of the string is assumed. The optional *count* argument specifies the number of substring substitutions to perform; if this is omitted, the default value is -1, which tells the function to make all possible substitutions. Finally, the optional *compare* is a numeric value indicating the kind of comparison to use when evaluating substrings and if this is omitted, the default value is 0 is assumed, which instructs the function to perform a binary comparison - the number for this last argument can be any value from Mode Compare.

The return value of the Replace function is a string in which the substitutions have been made.

Example

```
$Library "gfawinx"
Dim txt As String = "GFABasic32GFABasic32"
' Case insensitive replacement
Debug Replace(txt, "a", "xx", 4, 2, 1)
Debug.Show
```

The general instruction passed to the **Replace** function is to change all lower-case 'a' characters to 'xx'; however, this operation is expanded by the final *compare* argument of 1 which specifies that the search should be case INsensitive so all upper-case 'A' characters are to be replaced as well. Without any further arguments, this would result in four substitutions; however, the inclusion of the value 4 in the *start* argument means the first 'A' is excluded from the replacement process and the passing of the value 2 in the *count* argument means that only two substitutions are made, resulting in the final 'a' remaining unconverted. Hence, the output of this example is `GFABxxsic32GFxxBasic32`.

Remarks

Replace is defined using **FunctionVar** because this type takes implicit ByRef parameters (each parameter without an explicit ByVal is implicit ByRef). Consequently, when a literal string is passed – like “a” and “xx” - the compiler inserts code to copy the literal strings in hidden local variables that are then passed by reference. However, if the parameter is a string variable the variable is passed by reference without first making a copy.

See Also

reSub

{Created by Sjouke Hamstra; Last updated: 08/08/2019 by James Gaite}

DllVersion and DllVersion$ Functions, CommCtlVersion, ShellVersion Properties

Purpose

Return the version number of a DLL

Syntax

v! = **DllVersion**([fname])
v$ = **DllVersion$**([fname])

v! = *Screen*. **CommCtlVersion**
v! = *Screen*. **ShellVersion**

v!	*: single expression*
v$, fname	*: string expressions*

Description

DllVersion(*fname***)** returns the version number of a given DLL *fname*, if the DLL supports the version info function, otherwise it returns 0; if the DLL isn't located, the function returns -1. All values are returned as Single variables.

In a similar way, **DllVersion$(***fname***)** returns a string containing extended version information of a given DLL *fname*. When the DLL doesn't support the version info function, it returns an empty string, and when the DLL can't be found, the function returns "Error".

DllVersion() or **DllVersion("")** return the version number of the gfawin32.ocx runtime, while **DllVersion$()** or **DllVersion$("")** return the version information in the form of a string.

CommCtlVersion returns the DLL version number of CommCtl.dll, while **ShellVersion** returns the DLL version number of Shell32.dll.

Example

```
Trace DllVersion("shell32.dll")
// or, using the Screen object
Trace Screen.ShellVersion
// or using DllVersion$
Trace DllVersion$("shell32.dll")
Trace Screen.CommCtlVersion
Trace DllVersion()        // Prints the GFA runtime
  version...
Trace DllVersion$("")     // ...as does this.
Debug.Show
```

Known Issues

GFABASIC sometimes has problems processing comparisons involving **DllVersion** because **DllVersion** returns a floating point/single variable and all 'magic numbers' are assumed by GFABASIC to be double. The workaround for this is to place a ! after the comparison 'magic number' (to make it a single) as shown below:

```
// Run using GfaWin23 v2.32
Print DllVersion > 2.31          // Sometimes TRUE,
  sometimes not
Print DllVersion > 2.31!         // Always TRUE
```

[Reported by James Gaite, 23/10/2017; Solution updated by Sjouke Hamstra 05/11/2017]

{Created by Sjouke Hamstra; Last updated: 05/11/2017 by James Gaite}

PixelsPerTwipX, PixelsPerTwipY, TwipsPerPixelX, TwipsPerPixelY Properties

Purpose

Returns the number of pixels per twip or twips per pixel for **Screen**, **Form** and **Printer** object.

Syntax

# = *object*.**PixelsPerTwipX**
# = *object*.**PixelsPerTwipY**

# = *form*.**TwipsPerPixelX**
# = *form*.**TwipsPerPixelY**
# = **Printer**.**TwipPerPixelX**
# = **Printer**.**TwipPerPixelY**

object	*: Screen, Form, Printer*
form	*: Screen, Form*

Description

These properties return the number of pixels per twip or twips per pixel for the device of the **Screen**, a **Form,** or a **Printer**. For instance, for a **Printer** object, twips per pixel are 4.8 for 300dpi (1440/300), 2.4 for 600dpi (1440/600), 2.0 for 720 dpi (1440/720), 0.5 for 2880 dpi (1440/2880)

Note: For the **Printer** object the last two properties are **TwipPerPixelX** and **TwipPerPixelY** (the 's' after Twip is omitted) but they perform the same function.

Example

```
Debug.Show
OpenW 1
Trace Me.PixelsPerTwipX
Trace Me.PixelsPerTwipY
Trace Me.TwipsPerPixelX
Trace Me.TwipsPerPixelY
Trace Screen.PixelsPerTwipX
Trace Screen.PixelsPerTwipY
Trace Screen.TwipsPerPixelX
Trace Screen.TwipsPerPixelY
SetPrinterHDC Printer.hDC
Trace Printer.PixelsPerTwipX
Trace Printer.PixelsPerTwipY
Trace Printer.TwipPerPixelX
Trace Printer.TwipPerPixelY
```

Remarks

A form is part of the screen, and is actually a property of the **Screen** object (Screen.Forms), and thus returns the same values.

1 Twip (the base unit of GFA-BASIC 32 OLE) is 1/20 Point = 1 /1440 inch.

See Also

Screen, Form, Printer

{Created by Sjouke Hamstra; Last updated: 03/03/2018 by James Gaite}

RGBColor Command

Purpose

Sets the foreground and background color for graphic output.

Syntax

RGBColor fore [, back]

fore, back:integer expression

Description

RGBColor sets the foreground or background color (or both) using a RGB-Value. A RGB value is composed using the RGB function, which takes three byte values, each specifying a color tint red = 0..255, green = 0..255, and blue = 0..255.

When the system uses a palette the parameters specify a palette index.

Example

```
OpenW 1
Local b&, g&, i&, j&, r&
DefLine 6, 4
For i& = 0 To _X Step 4
  For j& = 0 To _Y Step 4
    r& = SinQ((i& + j&) / 4) * 127 + 128
    g& = SinQ(j& / 2) * 127 + 128
    b& = SinQ(i& / 2) * 127 + 128
    RGBColor RGB(r&, g&, b&)
    Line i&, j&, i&, j&
  Next j&
Next i&
DefLine 0, 1
```

Usage of the SysCol function:

```
OpenW 1
Local a%, b%
a% = SysCol(COLOR_BTNFACE)
b% = SysCol(COLOR_BTNTEXT)
RGBColor a%, b%
Text 10, 10, "HALLO GFA"
```

Remarks

Color internally uses **RGBColor**, so Color is a short form of **RGBColor**.

See Also

Color, QBColor, ForeColor, BkColor

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Automation

Introduction

Automation is a process by which one program may open, control and close a second through the use of a dedicated COM library. The purpose of such an action is to use the resources of that second program to execute a task and, if necessary, return one or more values to the calling program.

For the purpose of this article, we shall automate Internet Explorer as it is, currently, a program available to all Windows users. Other applications that can be automated include Word, Excel and Powerpoint and virtually any other program with an associated COM library.

Creating and Using Automated Objects

The principle of Automation is relatively simple.

To facilitate Automation you can either create a COM object using CreateObject, which causes an occurence of the called program to open in a new process, or use GetObject to try and take control of an occurence of the program which is already running.

Once you have access to the second program, you may then use its predefined Properties and Methods to navigate through the program, read, edit and extract content and perform whatever tasks specific to that application that the Methods allow.

Finally, once you have no further use for the called program, you can force it to exit (or leave it running if you wish) and then destroy the COM object.

Below is a very quick example of this process in practice. Firstly, a control window is drawn which allows for remote exiting of Internet Explorer; then the COM Object itself is created and used to start an occurence of Internet Explorer which, in turn, is used to navigate to Google to search for references to 'GFABasic'. Internet Explorer can be closed normally or remotely but the running program will not cease until the control window is closed.

```
// Draw Control Window
OpenW 1, 10, 10, 100, 100
Ocx Command cmd = "Close IE", 10, 10, (90 -
  (Screen.cxFrame * 2)), (90 - Screen.cyCaption -
  (2 * Screen.cyFrame))
// Create the COM Object
Global ie As Object
Set ie =
  CreateObject("InternetExplorer.Application")
If Not IsNothing(ie)
                    // If COM Object is created
  ie.Visible = True
                     // Show Internet Explorer
  ~ie.navigate("http://www.google.com/#hl=en&q=" &
    "GFABasic")  // Navigate to Google and Search
    for 'GFABasic'
  While ie.busy : DoEvents : Wend
                     // Wait for the page to finish
    loading
  While Not IsNothing(ie) : Sleep : Wend
                   // DoEvents until Internet Explorer
    or Control Window is closed
EndIf
```

```
CloseW 1
                    // Close Control Window

Sub cmd_Click
  If Not IsNothing(ie)
    // Try and close Internet Explorer
    Try
      ie.quit
    Catch
      // Internet Explorer has been closed by user
    EndCatch
    Set ie = Nothing
  EndIf
EndSub

Sub Win_1_Close(Cancel?)
  // Make sure Internet Explorer is closed and ie
    is Nothing
  cmd_Click
EndSub
```

It is beyond the scope of this article to list all the properties and methods that can be used but the following links may prove useful:

- MSDN - The Internet Explorer Object
- Automating Microsoft Office 97 and Microsoft Office 2000

_DispID and .{}

GFABasic has two functions which can be used when determining and returning values of Automation object properties: **_DispID** and **.{}**. These are discussed in detail on Sjouke Hamstra's blog but a quick precis of the details are as follows.

Generally, most properties related to a COM object can be accessed using the property's name: hence, to check that Internet Explorer is visible, `ie.Visible` is queried and returns either a True or False value accordingly.

Sometimes, a COM property may not be available to a program as a named property (usually because it is new): this is where **_DispID**(*Object*, *PropertyName*) and *Object***.{***IDispatchID***}** come in useful. **_DispID** queries the *Object* for the *PropertyName* which returns the *IDispatchID* (if the property exists) which can then be used by **.{}** to retrieve the relevant value. Hence, for the Internet Explorer *Visible* property, the following procedure could be followed:

```
Debug.Show
Dim ie As Object
Set ie =
 CreateObject("InternetExplorer.Application")
Trace ie.visible
Local idisp As Int32 = _DispID(ie, "Visible")
Trace idisp
Trace ie.{idisp}
~ie.quit
Set ie = Nothing
```

Unfortunately, at the time of writing, the **.{}** function can not be used to set the property due to a compiler error; when it is used in the logical fashion `ie.{402} = True`, an 'Internal: Set Prop value' error message is displayed.

{Created by Sjouke Hamstra; Last updated: 08/03/2018 by James Gaite}

AvailPhys, TotalPhys, AvailPageFile, TotalPageFile, AvailVirtual, TotalVirtual, MemoryLoad Properties (App)

Purpose

Return information about the physical and virtual memory size.

Syntax

*App.***AvailPhys**
*App.***TotalPhys**
*App.***AvailPageFile**
*App.***TotalPageFile**
*App.***AvailVirtual**
*App.***TotalVirtual**
*App.***MemoryLoad**

Return type:Long

Description

AvailPhys	Available physical global memory
TotalPhys	Total physical global memory
AvailPageFile	Available page file size
TotalPageFile	Total page file size
AvailVirtual	Available virtual memory

TotalVirtual	Total available virtual memory
MemoryLoad	Percentage of memory used.

Example

```
Debug.Show
Trace App.AvailPhys
Trace App.TotalPhys
Trace App.AvailPageFile
Trace App.TotalPageFile
Trace App.AvailVirtual
Trace App.TotalVirtual
Trace App.MemoryLoad
```

Known Issues

Similar to **mAlloc**(-1) through to **mAlloc**(-4), **AvailPageFile**, **AvailPhys**, **TotalPageFile** and **TotalPhys** are currently broken in most versions of Windows after XP SP3. See the mAlloc() page for the workaround.

Remarks

Same results can be obtained from **mAlloc**().

See Also

App, mAlloc

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

FileVersion Properties (App)

Purpose

Return the version information about the current running application.

Syntax

% = App.**Major**
% = App.**Minor**
% = App.**MajorRevision**
% = App.**Revision**
% = App.**ProdMajor**
% = App.**ProdMinor**
% = App.**ProdMajorRevision**
% = App.**ProdRevision**
$ = App.**Comments**
$ = App.**CompanyName**
$ = App.**FileDescription**
$ = App.**FileVersion**
$ = App.**InternalName**
$ = App.**LegalCopyright**
$ = App.**LegalTrademarks**
$ = App.**OriginalFilename**
$ = App.**PrivateBuild**
$ = App.**ProductName**
$ = App.**ProductVersion**
$ = App.**SpecialBuild**

Description

These properties return the file information as they are specified in the FileVersion tab of the Compile To Exe dialog box.

Properties	Meaning
Major	The major release number.
Minor	The minor release number.
MajorRevision	The major revision number.
Revision	The revision number.
ProdMajor	The major product version number.
ProdMinor	The minor product version number.
ProdMajorRevision	The major product revision number.
ProdRevision	The product revision number.
Comments	Comments
CompanyName	CompanyName
FileDescription	FileDescription
FileVersion	FileVersion
InternalName	Internal name
LegalCopyright	LegalCopyright
LegalTrademarks	Legal Trademarks
OriginalFilename	Original filename
PrivateBuild	Private build
ProductName	ProductName
ProductVersion	Product version

SpecialBuild	SpecialBuild

Example

```
Debug.Show
Trace App.Major
Trace App.Minor
Trace App.MajorRevision
Trace App.Revision
Trace App.ProdMajor
Trace App.ProdMinor
Trace App.ProdMajorRevision
Trace App.ProdRevision
Trace App.Comments
Trace App.CompanyName
Trace App.FileDescription
Trace App.FileVersion
Trace App.InternalName
Trace App.LegalCopyright
Trace App.LegalTrademarks
Trace App.OriginalFilename
Trace App.PrivateBuild
Trace App.ProductName
Trace App.ProductVersion
Trace App.SpecialBuild
```

See Also

App

{Created by Sjouke Hamstra; Last updated: 06/10/2014 by James Gaite}

FileName, Name, Path Properties (App)

Purpose

Return or set the filename and path of the current running application.

Syntax

$ = App.**FileName**

$ = App.**Name**

$ = App.**Path**

Description

FileName gets or sets (current) complete filename.

Name gets or sets (current) application name.

Path gets or sets (current) application path.

Example

```
Debug.Show
Trace App.FileName
Trace App.Name
Trace App.Path
```

Remarks

The complete path name can also be obtained from **_CmdLine**.

ProgName[$] returns the directory of the current running application. In the IDE the name of GFA-BASIC 32 is returned.

See Also

App, _CmdLine, ProgName

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

_Instance Variable, hInstance Property (App)

Purpose

Return the instance handle of the running program.

Syntax

Handle = **_Instance**

Handle = *App*.**hInstance**

Description

For 'interpreted' programs, the instance of the GFA-BASIC 32 IDE is returned. For a compiled program, the instance of the program is returned. Some Windows's API functions need this handle.

Example

```
Dim h As Handle
h = _INSTANCE : Print h
h = App.hInstance  : Print h
```

Remarks

Within GFA-BASIC 32 there are no interpreted programs, only compiled. However, when a program is run in the context of the IDE, the instance handle determines the GFA-BASIC 32 IDE.

See Also

App

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

scClear, scRead, scWrite, scPath, scDescription, scShowCmd, scDirectory, scArguments, scHotkey, scIconPath, scIconIndex (App)

Purpose

Properties and methods to manage shell links or shortcuts.

Syntax

App.**scClear**
App.**scRead(f$)**
App.**scWrite(f$)**

App.**scPath** [= *string*]
App.**scDescription** [= *string*]
App.**scShowCmd** [= *long*]
App.**scDirectory** [= *string*]
App.**scArguments** [= *string*]
App.**scHotkey** [= *long*]
App.**scIconPath** [= *string*]
App.**scIconIndex** [= *Long*]

Description

Usually, an user creates a shell link by choosing the Create Shortcut command from an object's context menu. The system automatically creates an icon for the shell link by combining the object's icon with a small arrow (known as the system-defined link overlay icon) that appears in the lower left corner of the icon. A shell link that has an icon is called a shortcut; however, the terms shell link and shortcut are often used interchangeably.

GFA-BASIC 32 applications can also create and use shell links and shortcuts. For example, a word processing application might create a shell link to implement a list of the most recently used documents. In GFA-BASIC 32 you create a shell link by using the methods and properties of the **App** object.

Methods:

The **scClear** method clears all **App.sc**XXX shell link related properties. These are the properties listed in the Syntax part.

The **scRead**(f$) method gets the settings from an external .lnk file f$.

The **scWrite**(f$) method creates and saves the shortcut file f$.

Properties:

scPath - Specifies the location (path) of the object referenced by the shortcut.

scDescription - Specifies the shortcut's description string, which the user never sees.

scShowCmd - Specifies the initial show state of the application (SW_xxx constant).

scDirectory - Specifies the working directory of the corresponding object, where files are loaded and saved when the user does not identify a specific directory.

scArguments - Specifies the arguments to pass to the object specified in **scPath**.

scHotkey - Global windows key to run the shortcut. The virtual key code is in the low-order byte, and the modifier flags are in the high-order byte. Bits 8, 9, and 10 represent Shift, Control, and Alt, respectively.

scIconPath - Icon file to use for the shortcut.

scIconIndex - Index to icon in the icon file specified in **scIconPath**.

The shortcut's name, which is a string that appears below the shell link icon, is actually the file name of the shortcut itself and which is specified in **scWrite**. The user can edit the description string by selecting it and entering a new string.

Example

```
App.scClear
App.scDirectory = SysDir
App.scIconIndex = 2
App.scIconPath = WinDir & "\Winfile.exe"
App.scArguments = #34 & SysDir & #34
App.scDescription = "System Folder"
App.scPath = WinDir & "\Explorer.Exe"
App.scWrite App.scPrograms & "\GFA32\System-
  Folder.lnk" // Change this to an appropriate file
  in your Program Files folder
```

See Also

App, scPrograms

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

scCommonStartMenu, scCommonPrograms, scStartMenu, scPrograms, scSpecialDir Properties

Purpose

Return the path of special folders.

Syntax

$ = App.**scCommonStartMenu**
$ = App.**scCommonPrograms**
$ = App.**scStartMenu**
$ = App.**scPrograms**
$ = App.**scSpecialDir(***csidl%***)**

Description

scCommonStartMenu returns the file system directory that contains the programs and folders that appear on the Start menu for all users (CSIDL_COMMON_STARTMENU).

scCommonPrograms returns the file system directory that contains the directories for the common program groups that appear on the Start menu for all users (CSIDL_COMMON_PROGRAMS).

scStartMenu returns the file system directory that corresponds to the user's Startup program group. The system starts these programs whenever any user logs onto Windows NT or starts Windows 95 (CSIDL_STARTMENU).

scPrograms returns the file system directory that contains the user's program groups, which are also file system directories (CSIDL_PROGRAMS).

scSpecialDir(*csidl%***)** returns the file system directory name for the special folders for the current user.

n	**ConstantDirectory**
0	CSIDL_DESKTOP\Documents and Settings\User\Desktop
2	CSIDL_PROGRAMS\Documents and Settings\User\Menu Start\Programs (**scPrograms**)
5	CSIDL_PERSONAL\Documents and Settings\User\My Documents
6	CSIDL_FAVORITES\Documents and Settings\User\Favourites
7	CSIDL_STARTUP\Documents and Settings\User\Menu Start\Programs\Start Up
8	CSIDL_RECENT\Documents and Settings\User\Recently Opened
9	CSIDL_SENDTO\Documents and Settings\User\SendTo
11	CSIDL_STARTMENU\Documents and Settings\User\Menu Start (**scStartMenu**)
16	CSIDL_DESKTOPDIRECTORY\Documents and Settings\User\Desktop
19	CSIDL_NETHOOD\Documents and Settings\User\NetHood
20	CSIDL_FONTS\Windows\Fonts
21	CSIDL_TEMPLATES\Documents and Settings\User\Templates
26	CSIDL_APPDATA\Documents and Settings\User\Application Data

27	CSIDL_PRINTHOOD\Documents and Settings\User\Networkprinters
32	CSIDL_INTERNET_CACHE\Documents and Settings\User\Local Settings\Temporary Internet Files
33	CSIDL_COOKIES\Documents and Settings\User\Cookies
34	CSIDL_HISTORY\Documents and Settings\User\Local Settings\History
35	CSIDL_COMMON_APPDATA\Documents and Settings\All Users\Application Data
36	CSIDL_WINDOWS\Windows (**WinDir$**)
37	CSIDL_SYSTEM\Windows\System32 (**SysDir$**)
38	CSIDL_PROGRAM_FILES\Program Files
39	CSIDL_MYPICTURES\Documents and Settings\User\My Documents\My Pictures
43	CSIDL_PROGRAM_FILES_COMMON\Program Files\Common Files
32768	CSIDL_FLAG_CREATE\Documents and Settings\User\Desktop
47	CSIDL_COMMON_ADMINTOOLS\Documents and Settings\All Users\Menu Start\Programs\Accessories\Admin
48	CSIDL_ADMINTOOLS\Documents and Settings\User\Menu Start\Programs\Admin

Example

```
Debug.Show
Trace App.scSpecialDir(CSIDL_DESKTOP)
Trace App.scSpecialDir(CSIDL_INTERNET)
Trace App.scSpecialDir(CSIDL_PROGRAMS)
Trace App.scSpecialDir(CSIDL_CONTROLS)
Trace App.scSpecialDir(CSIDL_PRINTERS)
Trace App.scSpecialDir(CSIDL_PERSONAL)
Trace App.scSpecialDir(CSIDL_FAVORITES)
Trace App.scSpecialDir(CSIDL_STARTUP)
Trace App.scSpecialDir(CSIDL_RECENT)
Trace App.scSpecialDir(CSIDL_SENDTO)
Trace App.scSpecialDir(CSIDL_BITBUCKET)
Trace App.scSpecialDir(CSIDL_STARTMENU)
Trace App.scSpecialDir(CSIDL_DESKTOPDIRECTORY)
Trace App.scSpecialDir(CSIDL_DRIVES)
Trace App.scSpecialDir(CSIDL_NETWORK)
Trace App.scSpecialDir(CSIDL_NETHOOD)
Trace App.scSpecialDir(CSIDL_FONTS)
Trace App.scSpecialDir(CSIDL_TEMPLATES)
Trace App.scSpecialDir(CSIDL_COMMON_STARTMENU)
Trace App.scSpecialDir(CSIDL_COMMON_PROGRAMS)
Trace App.scSpecialDir(CSIDL_COMMON_STARTUP)
Trace
  App.scSpecialDir(CSIDL_COMMON_DESKTOPDIRECTORY)
Trace App.scSpecialDir(CSIDL_APPDATA)
Trace App.scSpecialDir(CSIDL_PRINTHOOD)
Trace App.scSpecialDir(CSIDL_ALTSTARTUP)
Trace App.scSpecialDir(CSIDL_COMMON_ALTSTARTUP)
Trace App.scSpecialDir(CSIDL_COMMON_FAVORITES)
Trace App.scSpecialDir(CSIDL_INTERNET_CACHE)
Trace App.scSpecialDir(CSIDL_COOKIES)
Trace App.scSpecialDir(CSIDL_HISTORY)
Debug.Print
Trace App.scPrograms     // CSIDL_PROGRAMS
Trace App.scStartMenu    // CSIDL_STARTMENU
Do
  Sleep
Until Me Is Nothing

Sub CSIDL
  Global Const CSIDL_DESKTOP       =
    0x0000
```

```
Global Const CSIDL_INTERNET      =
  0x0001
Global Const CSIDL_PROGRAMS      =
  0x0002
Global Const CSIDL_CONTROLS      =
  0x0003
Global Const CSIDL_PRINTERS      =
  0x0004
Global Const CSIDL_PERSONAL      =
  0x0005
Global Const CSIDL_FAVORITES     =
  0x0006
Global Const CSIDL_STARTUP       =
  0x0007
Global Const CSIDL_RECENT        =
  0x0008
Global Const CSIDL_SENDTO        =
  0x0009
Global Const CSIDL_BITBUCKET     =
  0x000a
Global Const CSIDL_STARTMENU     =
  0x000b
Global Const CSIDL_DESKTOPDIRECTORY  =
  0x0010
Global Const CSIDL_DRIVES            =
  0x0011
Global Const CSIDL_NETWORK           =
  0x0012
Global Const CSIDL_NETHOOD           =
  0x0013
Global Const CSIDL_FONTS             =
  0x0014
Global Const CSIDL_TEMPLATES         =
  0x0015
Global Const CSIDL_COMMON_STARTMENU  =
  0x0016
Global Const CSIDL_COMMON_PROGRAMS    =
  0X0017
Global Const CSIDL_COMMON_STARTUP     =
  0x0018
Global Const CSIDL_COMMON_DESKTOPDIRECTORY =
  0x0019
Global Const CSIDL_APPDATA                =
  0x001a
Global Const CSIDL_PRINTHOOD              =
  0x001b
Global Const CSIDL_ALTSTARTUP              =
  0x001d          // DBCS
Global Const CSIDL_COMMON_ALTSTARTUP       =
  0x001e          // DBCS
Global Const CSIDL_COMMON_FAVORITES        =
  0x001f
Global Const CSIDL_INTERNET_CACHE          =
  0x0020
Global Const CSIDL_COOKIES                 =
  0x0021
Global Const CSIDL_HISTORY                 =
  0x0022
```

Remarks

scSpecialDir doesn't return the path for all CSIDL constants. For those not supported, an API call must be invoked.

```
Dim Path$ = Space(260)
If SHGetFolderPath(0, csidl%, 0, 0, Path$) = 0
  Path$ = ZTrim(Path$)
EndIf
Declare Function SHGetFolderPath Lib "shell32.dll"
  Alias "SHGetFolderPathA" (ByVal hwnd As Long,
  ByVal csidl As Long, ByVal hToken As Long, ByVal
  dwFlags As Long, ByVal pszPath As String) As Long
```

See Also

App

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

_Name$ and ComputerName, UserName, WinCompany, WinUser Properties (App)

Purpose

Returns the name of the client PC and the login name.

Syntax

$ = **_Name**$

$ = App.**ComputerName**
$ = App.**UserName**
$ = App.**WinCompany**
$ = App.**WinUser**

Description

Returns the name of the PC as it is known on a LAN. Without a network installation **_Name**$ returns an empty string. The App.**ComputerName** property is identical.

App.**UserName** returns the user that is currently logged on to the system.

The **App** properties **WinCompany** and **WinUser** return the company and user owning the Windows system.

Example

```
Print "The name of the PC is: "; _Name$
Print "The App name of the PC is: ";
 App.ComputerName
Print "The App login name is: "; App.UserName
Print "The Company owning Windows is: ";
 App.WinCompany
Print "The User owning Windows is: "; App.WinUser
```

See Also

App Object

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

FontCount, Fonts Properties (Screen, Printer)

Purpose

Returns the number of fonts, and all names, available for the current display device or active printer.

Syntax

% = *object*.**FontCount**

$ = *object*.**Fonts**(i%)

object:Screen, Printer Ocx

Description

The **Fonts** property works in conjunction with the **FontCount** property, which returns the number of font names available for the object. The parameter i% is an integer from 0 to **FontCount** -1.

Example

```
Dim i%
Debug.Show
Debug.Print "PRINTER FONTS" : Debug
For i = 0 To Printer.FontCount - 1
  Debug.Print Printer.Fonts(i)
Next
Debug : Debug.Print "SCREEN FONTS" : Debug
For i = 0 To Screen.FontCount - 1
  Debug.Print Screen.Fonts(i)
```

```
Next
```

Remarks

Fonts available vary according to your system configuration, display devices, and printing devices.

See Also

Printer, Screen

{Created by Sjouke Hamstra; Last updated: 06/10/2014 by James Gaite}

hWnd Property

Purpose

Returns a handle to a form or control.

Syntax

h = *object*.**hWnd**

object:Ocx Object
h:Handle

Description

The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or **hWnd**. The **hWnd** property is used with Windows API calls. Many Windows operating environment functions require the **hWnd** of the active window as an argument.

For the **Screen** object **hWnd** returns the handle of the desktop window.

Example

```
OpenW 1
Print Win_1.hWnd
```

Remarks

Because the value of this property can change while a program is running, never store the **hWnd** value in a variable.

See Also

hDC, hDC2

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

WinVersion Function, WinVer Property

Purpose

Return a string specifying the running Windows version.

Syntax

$ = **WinVersion**

$ = *Screen*.**WinVer**

Description

WinVersion and **WinVer** should both return the version of the currently running windows. **WinVersion** doesn't work correctly on XP.

Example

```
Debug.Show
Trace WinVersion
Trace Screen.WinVer
```

Remarks

-

See Also

Screen, WinVer Function

{Created by Sjouke Hamstra; Last updated: 13/08/2019 by James Gaite}

Screen_KeyPreview Event

Purpose

Intercepts keyboard events before they are dispatched to the target form or control.

Syntax

Sub Screen_KeyPreview(hWnd%, uMsg%, wParam%, lParam%, Cancel?)

Description

The event is invoked before GFA-BASIC 32 or Windows handles the key press. The event provides a central place to filter key events or to handle keyboard events in a custom manner (to create an editor, or something similar).

You can use this event to create a keyboard-handling procedure for an application. For example, when an application uses function keys, you'll want to process the keystrokes at the application level rather than writing code for each form or control that might receive keystroke events.

The event parameters:

hWnd%	Windows handle of the message
uMsg%	The window message number (WM_CHAR, WM_DEADCHAR, WM_KEYDOWN, WM_KEYUP, WM_SYSCHAR, WM_SYSDEADCHAR, WM_SYSKEYDOWN or WM_SYSKEYUP).
wParam%	The key code or ASCII value, dependant upon

	the window message (see Key Codes and ASCII Values).
lParam%	Extended key information dependant upon the window message.
Cancel?	Return value. Set to True when the keyboard message is to be ignored.

Example

```
Debug.Show
OpenW 1, 0, 0, 200, 200
Do
  Sleep
Until Win_1 Is Nothing

Sub Screen_KeyPreview(hWnd%, Msg%, wParam%,
  lParam%, Cancel?)
  ' Display keyboard message values.
  ' Don't use a MsgBox or the like.
  Debug Hex(hWnd) ` Hex(Msg) ` Hex(wParam) `
    Hex(lParam)
  ' Determine the (child) window the message is
    for.
  If hWnd = Win_1.hWnd
    ' Do your thing
    If Msg = WM_KEYDOWN && wParam = VK_F1
      Debug "Filtering F1 for Win_1"
      Cancel? = True
    EndIf
    ' or alternative:
  Else If Form(hWnd) Is Win_1
  Else
    Local Object Obj
    Set Obj = OCX(hWnd)
    ' Test for Nothing or use Try/Catch
    If IsNothing(Obj) Then
```

```
      Exit Sub
    EndIf
    ' Filter TAB for all TextBoxes (Example)
    If TypeOf(Obj) Is TextBox && _
      Msg = WM_KEYDOWN && wParam = VK_TAB
      Cancel? = True
    EndIf
    ' Test for a message for a child control
    If Obj.Parent Is Win_1
      Debug "ChildOcx of Win_1"
    End If
  EndIf
End Sub
```

If the control receiving the key press is a ComboBox, the return value for **Typename**(**OCX**(*hWnd%*)) in this example will return 'Nothing' as the handle refers to the edit box within the ComboBox; to get the actual ComboBox, use **OCX**(**GetParent**(*hWnd%*)) instead. However, if **OCX**() produces a recognised type, **GetParent**() can invoke a Type Error so it is advisable to use the *obj*.**Parent** property in that instance.

Remarks

Don't use a **MsgBox**, **Alert**, **Input** or some other dialog box inside the Screen_KeyPreview event sub. All keyboard events arrive in this sub, so that a recursive call of Screen_KeyPreview is more than likely.

Also, don't respond to messages in this event sub. The only thing that is allowed is to cancel a keyboard message by setting Cancel? to True. SetFocus is allowed, but only with one of the corresponding messages, for instance with WM_KEYDOWN, but not with WM_KEYUP, or the other way round.

When the Shift, Control, Alt and/or Alt Gr keys are pressed, a new event is created; however, any charcter keys entered while they are pressed do not always carry details of the state of these keys. It is possible to set up **Static** booleans to track the shift key state although it is more reliable to query the **Screen**.**ShiftKeys** property when necessary.

See Also

Screen, KeyPress, KeyUp, KeyDown

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Number, Description, Exception Properties

Purpose

Return or set a numeric value and a short description of an error.

Syntax

Err.**Number** [= integer]

Err.**Description** [= string]

Err.**Exception**

Description

In case of a runtime error, GFA-BASIC 32 returns the value of the error in the **Number** property. The **Description** property returns a short description of the error. The numbers 0 to 141 are reserved for GFA-BASIC 32 errors. Other numbers can be used to generate user-defined errors. A list of error numbers and description are found here.

Number is the **Err** object's default property. **Err** is identical to **Err**.**Number**.

When **Err** = 46 an OLE object is the source of the error and you must inspect the **HResult** property for more details.

When **Err** is between 93 and 115, a system hardware or system software problem is responsible for the error. These events normally terminate program execution. Such events are called exceptions, and the mechanism that deals with exceptions is called structured exception handling. Normally, GFA-BASIC 32 reports these runtime errors with a message box and then terminates the program. By using *structured exception handling* you can handle both hardware and software exceptions. Therefore, your code will handle hardware and software exceptions identically. GFA-BASIC 32 supports structured exception handling with the **Try**/**Catch**/**EndCatch** statements. GFA-BASIC 32 translates the exception code into one of its own error numbers (93 to 115), but you can use the **Exception** property to retrieve a code that identifies the reason for the exception. For instance, an "Object is Nothing" error might be the result of an access violation, in which case **Exception** = $c0000005.

Inspecting and handling an error condition is only possible when the error is trapped. The preferred way in GFA-BASIC 32 is by using **Try**/**Catch**/**EndCatch**. However for compatibility reasons the VB structure **On Error Goto** can be used as well.

Example

```
Debug.Show
Local a
Try
  Monitor ' a breakpoint
Catch
  Debug Err.Number, Err$
  Debug Hex(Err.Exception)
EndCatch
Try
  a = (1 \ 0)   ' integer division
Catch
  Debug Err.Number, Err$
  Debug Hex(Err.Exception)
```

```
EndCatch
Try
  a = (1 / 0)   ' floating point division
Catch
  Debug Err.Number, Err$
  Debug Hex(Err.Exception)
EndCatch
```

Try the all three error conditions by commenting the different lines. They are all the result of an exception.

See Also

Err Object, Err$, Try

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

HelpContext, HelpFile, Source Properties (Err)

Purpose

Return or set the help properties for the **Err** object.

Syntax

Err.**HelpContext** [= long]

Err.**HelpFile** [= string]

Err.**Source** [= string]

Description

The **HelpContext, HelpFile,** and **Source** properties are set in conjunction with the **Raise** method. The **Raise** method allows you to create and generate user-defined errors for your application. When the application error is described in a help file, the context ID and the name of the file can be specified in these properties or as parameters in the **Raise** method.

If you are calling an older WinHlp32 (.hlp) help file, then the MsgBox statement is particularly useful to present the error information to the end user and if you a calling a newer HTMLHelp file type, a different message box structure detailed here can perform the same task. Alternatively, you can create your own display or message box using the Dialog object.

The **Source** property returns or sets a string specifying the name of the object or application that originally generated the error. For GFA-BASIC 32 runtime errors it is "GFA-BASIC 32", for OLE Automation errors it is the COM program name. When generating an error from code, **Source** is your application’s program name.

Example

You can use the following code to launch a message box:

```
Try
  Err.Raise 1, "Quick Example", "This is a
    manufactured error", "HelpFilepath", 25
Catch
  ~MsgBox(Err.Description & #13#10#13#10 & "Help
    File: " & Err.HelpFile & #13#10 & "Help
    Context:" & Err.HelpContext, MB_OK, Err.Source)
EndCatch
```

Remarks

The **Source** property specifies a string expression representing the object that generated the error; the expression is usually the object's class name, program name, or programmatic ID. Use **Source** to provide information when your code is unable to handle an error generated in an accessed object. For example, if you access Microsoft Excel and it generates a Division by zero error, Microsoft Excel sets Err.**Number** to its error code for that error and sets **Source** to Excel.Application.

See Also

Err object, Raise, Err$

{Created by Sjouke Hamstra; Last updated: 17/07/2015 by James Gaite}

LastDLLError Property

Purpose

Returns the last error code of a Win32 function.

Syntax

% = *Err*.**LastDLLError**

Description

The **LastDLLError** property invokes the *GetLastError* API function to obtain the calling thread's last-error code value. Note that the **LastDLLError** is read-only.

When an error occurs, most Win32 functions return an error code, usually False, Null, 0xFFFFFFFF, or -1. Many functions also set an internal error code called the *last-error code*. When a function succeeds, the last-error code is not reset. The error code is maintained separately for each running thread; an error in one thread does not overwrite the last-error code in another thread. An application can retrieve the last-error code by using the *GetLastError* function; the error code may tell more about what actually occurred to make the function fail.

You should call the *GetLastError* function immediately when a function's return value indicates that such a call will return useful data. That is because some functions call *SetLastError*(0) when they succeed, wiping out the error code set by the most recently failed function.

Whenever the failure code is returned, the GFA-BASIC 32 application should immediately check the **LastDLLError** property. No exception is raised when the **LastDLLError** property is set.

To obtain an error string for system error codes, use the **SysErr$** function and pas the value returned by **LastDLLError**.

Example

```
' The following lines behave identically
Print SysErr(Err.LastDllError)
Print SysErr(GetLastError())
```

Remarks

Some (older) VB documentation wrongly suggest that **LastDLLError** contains the return value of the last invoked Win32 API function. This is untrue both in VB and GFA-BASIC 32.

See Also

Err Object, SysErr

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

HResult Property (Err)

Purpose

Return value of COM methods and properties.

Syntax

Err.**HResult**

Description

An HRESULT is defined in COM as a result handle that can be used for determining the success of failure of a function. However, rather than a handle the HRESULT returns a 32-bits error value. The HRESULT value of properties and methods is examined by GFA-BASIC 32 and in case of failure GFA-BASIC 32 generates an error. GFA-BASIC 32 stores the COM (automation) error in the **HResult** property of the **Err** object.

The errors are mostly between $80040000 and $8004FFFF, but not always. Even for automation objects that don't confirm to MS convention (MS Word), GFA-BASIC 32 sets the high word to $8004 to provide a way to identify object errors and normal GFA-BASIC 32 errors. In addition, in case of an OLE-error, the GFA-BASIC 32 documentation says, that the **Number** property is set to 46 (= Error with object). This might be true (?) for errors with late-binding objects (Object data type), but certainly not with the GFA-BASIC 32 Ocx objects, as we will see.

When an error trap (**Try**/**Catch** or **On Error GoTo**) wants to differentiate between the a BASIC error and a COM error, the best way to go is by checking for a non-null value in **HResult**.

If Err.HResult Then OLE-error **Else** GFA-BASIC 32 - error

The HRESULT value can be used to determine the cause of the COM error. The lower word (16-bits) describes what actually took place, error or otherwise. Bits 15 to 30 indicate to which group of status codes the HRESULT value belongs, the so called facility codes. Actually, the facility codes range from 0 to 10 and can be obtained by reading the second byte using **GetByte1**(). See Remarks for a list of facility codes.

Example

The following example invokes the **Node** method **CreateDragImage**, which isn't implemented so far. The returned HRESULT value is examined by GFA-BASIC 32 and an error is generated and trapped. Both, **HResult** and **Number,** contain the value E_NOTIMPL ($80004001). The **Description** property returns a user-readable message for the HRESULT, localized to the user's language as appropriate.

```
Print
Dim node As Node, p As Picture
Ocx TreeView tv = "", 250, 10, 230, 200
tv.Add , , , "Painters"
Set node = tv.Add(1, tvwChild, "David" , "David")
' invoke a non-implemented method
Try
  Set p = node.CreateDragImage
Catch
  Debug.Show
  Debug Hex(Err.HResult)
  Debug Hex(Err), Err.Description
  Debug "Facility code: ";GetByte1(Err.HResult)
EndCatch
```

The **Try** statement resets the **Err** object to default values (0 and "").

Note - Actually, the missing CreateDragImage method doesn't generate an error itself like a BASIC error or exception is generated. The method of the Node object is available and the program actually gets inside the CreateDragImage function. However, the COM method immediately returns with E_NOTIMPL, which becomes the HRESULT return value for the method. It is up to the client of CreateDragImage how to handle the return value. By default, GFA-BASIC 32 generates an error. After each call of a property or method GFA-BASIC 32 inserts code to check the return value (HRESULT). If it is not S_OK (0) the **Err** object is filled and the program comes to a halt. This behavior can be disabled by putting **$ObjNoErr** into the code.

Remarks

The following table lists the values of common HRESULT values. (These constants are not implemented in GFA-BASIC 32.)

Name	**Value**	**Description**
S_OK	0x00000000	Operation successful
E_UNEXPECTED	0x8000FFFF	Unexpected failure
E_NOTIMPL	0x80004001	Not implemented
E_OUTOFMEMORY	0x8007000E	Failed to allocate necessary memory
E_INVALIDARG	0x80070057	One or more arguments are invalid

E_NOINTERFACE	0x80004002	No such interface supported
E_POINTER	0x80004003	Invalid pointer
E_HANDLE	0x80070006	Invalid handle
E_ABORT	0x80004004	Operation aborted
E_FAIL	0x80004005	Unspecified failure
E_ACCESSDENIED	0x80070005	General access denied error
E_NOTIMPL	0x80000001	Not implemented

The following table describes the various facility fields:

FACILITY_NULL (0) - For broadly applicable common status codes such as S_OK.

FACILITY_RPC (1) - For status codes returned from remote procedure calls.

FACILITY_DISPATCH (2) - For late-binding IDispatch interface errors.

FACILITY_STORAGE (3) - For status codes returned from **IStorage** or **IStream** method calls relating to structured storage. Status codes whose code (lower 16 bits) value is in the range of DOS error codes (that is, less than 256) have the same meaning as the corresponding DOS error.

FACILITY_ITF (4) - Most commonly specified code, returned from interface methods, value is defined by the interface.

FACILITY_WIN32 (7) - Used to provide a means of handling error codes from functions in the Win32 API as an HRESULT.

FACILITY_WINDOWS (8) - Used for additional error codes from Microsoft-defined interfaces.

FACILITY_CONTROL (10) - Result related to OLE controls.

Note that a number of HRESULT codes are related to Win32 API functions, because the facility code is 7.

```
If GetByte1(Err.HResult) = 10   ' an Ocx related
 error
```

For more information on COM see the MS SDK.

See Also

Err Object

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

CancelError Property (CommDlg)

Purpose

Returns or sets a value indicating whether an error is generated when the user chooses the Cancel button.

Syntax

CommDlg.**CancelError** [= *Bool*]

Description

To prevent errors from occurring in your application, such as specifying a nonexistent color in the Color dialog box, you can use the **CancelError** property. This property lets you know if the user clicked the Cancel button on the dialog box. Each of the six dialog boxes uses the **CancelError** property. The **CancelError** property lets you set a trap (Try/Catch) for the Cancel button. When this property is set to True, GFA-BASIC 32 generates an error (CDERR_CANCEL or 32755) that you can trap in your program. If **CancelError** is set to False, no error occurs-the dialog box simply closes and returns a NULL value.

Example

```
Global Enum CC_RGBINIT =1, CC_FULLOPEN
OpenW Hidden 1
Ocx CommDlg cd
cd.CancelError = True
cd.Flags = CC_RGBINIT Or CC_FULLOPEN
```

```
Try
  cd.ShowColor
Catch
  Message "Cancel clicked"
EndCatch
CloseW 1
```

See Also

CommDlg

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

ShowColor Method, Color, Colors

Purpose

Displays the **CommDlg** control's Color dialog box.

Syntax

CommDlg.**ShowColor**

CommDlg.**Color** [= *rgb%*]
CommDlg.**Colors(**0..15**)** [= *rgb%*]

Description

The **Color** property returns or sets the selected color.

If the **cdcRgbInit** flag is set, the value set with **Color** specifies the color initially selected when the dialog box is created. If the specified color value is not among the available colors, the system selects the nearest solid color available. If **Color** is zero or **cdcRgbInit** is not set, the initially selected color is black. If the user clicks the OK button, **Color** specifies the user's color selection.

The **Colors**(0..15) property is an array of integers of 16 RGB color values that contain red, green, blue (RGB) values for the custom color boxes in the dialog box.

The **Flags** property can be used to set the options for a Color dialog box.

cdcFullOpen (2)	Entire dialog box is displayed,

	including the Define Custom Colors section.
cdcShowHelp (8)	Causes the dialog box to display a Help button.
cdcPreventFullOpen (4)	Disables the Define Custom Colors command button and prevents the user from defining custom colors.
cdcRgbInit (1)	Sets the initial color value for the dialog box.
cdcSolidColor (128)	Causes the dialog box to display only solid colors in the set of basic colors.
cdcAnyColor (256)	Causes the dialog box to display all available colors in the set of basic colors.

Example

```
Print
Ocx CommDlg cd
Dim i As Int
For i = 0 To 15
  cd.Colors(i) = QBColor(i)   //custom colors
Next
cd.Color = colBtnFace
cd.Flags = cdcRgbInit | cdcShowHelp
cd.CancelError = True
cd.ShowColor

Sub cd_OnHelp
  Me.Caption = "Help Requested"
EndSub
```

EndSub

See Also

CommDlg, Dlg Color

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

ShowOpen, ShowSave Methods, FileName, IniDir, FileTitle, DefExt, Filter, FilterIndex, Title Properties

Purpose

Display the **CommDlg** control's Open and Save As dialog box.

Syntax

CommDlg.**ShowOpen**
CommDlg.**ShowSave**

CommDlg.**FileName** [= *string*]
CommDlg.**IniDir** [= *string*]
CommDlg.**FileTitle** [= *string*]
CommDlg.**DefExt** [= *string*]
CommDlg.**Filter** [= *string*]
CommDlg.**FilterIndex** [= *integer*]
CommDlg.**Title** [= *string*]

Description

The **FileName** property returns or sets the path and filename of a selected file. In the **CommDlg** object, you can set the **FileName** property before opening a dialog box to set an initial filename.

The **IniDir** property is used to specify the initial directory for an Open or Save As dialog. If this property isn't specified, the current directory is used.

FileTitle returns or sets the name (without the path) of the file to open or save.

DefExt returns or sets the default filename extension for the dialog box, such as .txt or .doc. When a file with no extension is saved, the extension specified by this property is automatically appended to the filename.

Filter specifies the type of files that are displayed in the dialog box's file list box. For example, selecting the filter *.txt displays all text files. Use the pipe (|) symbol (ASCII 124) to separate the *description* and *filter* values. Don't include spaces before or after the pipe symbol, because these spaces will be displayed with the *description* and *filter* values. The following code shows an example of a filter that enables the user to select text files or graphic files that include bitmaps and icons:

```
Text (*.txt)|*.txt|Pictures (*.bmp;*.ico)|*.bmp;*.ico
```

When you specify more than one filter for an Open or Save As dialog box, use the **FilterIndex** property to determine which filter is displayed as the default. The index for the first defined filter is 1.

The **Title** property returns or sets the string displayed in the title bar of the dialog box. The default title for an Open dialog box is Open; the default title for a Save As dialog box is Save As.

The **Flags** property for **ShowOpen** and **ShowSave** can be:

cdoReadOnly $1	Causes the Read Only check box to be initially checked when the dialog box is

	created. This flag also indicates the state of the Read Only check box when the dialog box is closed.
cdoOverwritePrompt $2	Causes the Save As dialog box to generate a message box if the selected file already exists. The user must confirm whether to overwrite the file.
cdoHideReadOnly $4	Hides the Read Only check box.
cdoNoChangeDir $8	Forces the dialog box to set the current directory to what it was when the dialog box was opened.
cdoShowHelp $10	Displays the Help button.
cdoNoValidate $100	Allows invalid characters in the returned **Filename**.
cdoAllowMultiselect &H200	Allows multiple selections. The user can select more than one file at run time by pressing the SHIFT key and using the UP ARROW and DOWN ARROW keys to select the desired files. When this is done, the **FileName** property returns a string containing the names of all selected files. The names in the string are delimited by spaces.
cdoExtensionDifferent $400	Indicates that the extension is different from **DefExt**

	property.
cdoPathMustExist $800	Only valid paths. If the user enters an invalid path, a warning message is displayed.
cdoFileMustExist $1000	Allows only names of existing files. If the user enters an invalid filename, a warning is displayed. This flag automatically sets the **cdoPathMustExist** flag.
cdoCreatePrompt $2000	Prompts the user to create a file that doesn't currently exist. This flag automatically sets the **cdoPathMustExist** and **cdoFileMustExist** flags.
cdoNoReadOnlyReturn $8000	The returned file won't have the Read Only attribute set and won't be in a write-protected directory.
cdoNoTestFileCreate $10000	Specifies that the file is not created before the dialog box is closed. This flag should be specified if the application saves the file on a create-non-modify network share. When an application specifies this flag, the library does not check for write protection, a full disk, an open drive door, or network protection. Applications using this flag must perform file operations carefully, because a file

	cannot be reopened once it is closed.
cdoNoNetworkButton $20000	Hides and disables the **Network** button.
cdoNoLongNames $40000	No long file names.
cdoExplorer $80000	Use the Explorer-like Open A File dialog box template. Works with Windows 95 and Windows NT 4.0.
cdoNorefLinks $100000	Do not dereference shell links (also known as shortcuts). By default, choosing a shell link causes it to be dereferenced by the shell.

Example

```
Ocx CommDlg cd
cd.FileName = "test.file"
cd.FileTitle = ""
cd.DefExt = "FILE"
cd.Filter = "Files (*.FILE)|*.FILE|All Files
  (*)|*"
cd.Title = "Select a File"
cd.Flags = cdoHideReadOnly + cdoFileMustExist
cd.ShowOpen
Message cd.FileName
```

See Also

CommDlg, Dlg Open, Dlg Save

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

ShowPrint Method, Copies, FromPage, ToPage, Min, Max, hDC, DevNames

Purpose

Creates a **Printer** dialog box that enables the user to specify the attributes of a printed page. These attributes include the paper size and source, the page orientation (portrait or landscape), and the width of the page margins.

Syntax

CommDlg.**ShowPrint**

CommDlg.**Copies** [= *long*] *CommDlg*.**FromPage** [= *long*]
CommDlg.**ToPage** [= *long*]
CommDlg.**Min** [= *long*]
CommDlg.**Max** [= *long*]
CommDlg.**hDC** [= *long*]

CommDlg.**DevNames** [= *string*]

Description

The **ShowPrint** common dialog box allows the user to choose any printer and change the various settings. The controls of the dialog box are initialized using the associated **CommDlg** properties.

The **Copies** property specifies the initial number of copies to print.

The **FromPage** property specifies the page to start printing and the **ToPage** property the page to stop printing.

The **Min** and **Max** properties return or set the minimum and maximum allowed values for the print range.

The **hDC** property returns a device context for the printer selected in the Print dialog box when the **cdpReturnDC** flag is set or an information context when the **cdpReturnIC** flag is set.

The DevNames property returns the selected printer as a string with, comma delimited, Driver, Device, and Output Port. For example "WINSPOOL,HP Laserjet 4,LPT1:".

The **Flags** property values for **ShowPrint** indicate what services are requested in the dialog box.

Flags	**Meaning**
cdpAllPages $0	The default flag that indicates that the **All** radio button is initially selected.
cdpSelection $1	The **Selection** radio button is selected.
cdpPageNums $2	The **Pages** radio button is selected.
cdpNoSelection $4	Disables the **Selection** radio button.
cdpNoPageNums $8	Disables the **Pages** radio button and the associated edit controls.
cdpCollate $10	The **Collate** check box is checked.
cdpPrintToFile $20	The **Print to File** check box is selected.

cdpPrintSetup $40	Display the **Print Setup** dialog box rather than the **Print** dialog box.
cdpNoWarning $80	Prevents the system from displaying a warning message when there is no default printer.
cdpReturnDC $100	Returns a device context matching the selections the user made in the dialog box. The device context is returned in **hDC**.
cdpReturnIC $200	Returns an information context matching the selections the user made in the dialog box. The device context is returned in **hDC**.
cdpReturnDefault $400	Returns the standard printer in **DevNames** without showing the dialog box.
cdpShowHelp $800	Displays the **Help** button
cdpUseDevmodeCopies $40000	Indicates whether your application supports multiple copies and collation.
cdpDisablePrintToFile $80000	Disables the **Print to File** check box.
cdpHidePrintToFile $100000	Hides the **Print to File** check box.
cdpNoNetworkButton $2000000	Hides and disables the **Network** button.

This dialog box does not send data to the printer but lets the user specify how they want data printed. The following properties contain information about the user's selection: **Copies**, **FromPage**, and **ToPage**.

The printer device settings selected using the dialog box are made active when you make that printer the default printer for the application. This is accomplished by setting the **Printer** object to the CommDlg object.

Example

```
OpenW 1
Ocx CommDlg cd
cd.Flags = 0
cd.ShowPrint' change printer settings
' the user wants:
Trace cd.Copies
Trace cd.FromPage
Trace cd.ToPage
Try
  Set Printer = cd ' initialize the Printer object
  Trace Printer.hDC
  Trace Printer.Width
  Trace Printer.DeviceName
  Trace Printer.Orientation
  Trace Printer.dmPaperSize
Catch
  // Printer not set
EndCatch
CloseW 1
Debug.Show
```

Remarks

The device mode settings for the printer are stored in the DEVMODE structure, which is a shared object between the **ShowPrint** and **ShowPageSetup** dialog box. The **Printer** object needs to be initialized with the DEVMODE structure before it can be changed using **Printer** properties. By default, the **Printer** object is initialized with the device mode settings from the Windows standard printer. When you change the DEVMODE structure through the use of the **ShowPrint** (or **ShowPageSetup)** dialog box, the **Printer** object needs to be re-initialized. This is accomplished by assigning the **CommDlg** object to the **Printer** object using **Set**.

Known Issues

In some builds of GB32, the 'Pages..to..from' section is disabled regardless of whether flag cdpPageNums is set or not; if you require this function, use Dlg Print instead.

See Also

CommDlg, ShowPageSetup, Dlg Print, dm-Properties

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Enabled Property

Purpose

Returns or sets a value that determines whether a form or control can respond to user-generated events.

Syntax

*Object***.Enabled** [= Boolean]

Object:Ocx object
Boolean:True or False

Description

The **Enabled** property allows forms and controls to be enabled or disabled at run time. For example, you can disable objects that don't apply to the current state of the application. You can also disable a control used purely for display purposes, such as a text box that provides read-only information.

The default setting is True, which allows *object* to respond to events. Setting it to False prevents it from responding to events.

Disabling a **Timer** control by setting **Enabled** to **False** cancels the countdown set up by the control's **Interval** property.

For a **MenuItem** object, **Enabled** is normally read/write at run time.

Example

```
Form frm1
Ocx TextBox txt1 = "", 10, 10, 100, 14 :
  .BorderStyle = 1
Ocx Command cmd1 = "Save", 20, 35, 80, 22 :
  cmd1.Enabled = False
txt1.SetFocus
Do
  Sleep
Until Me Is Nothing

Sub cmd1_Click
  frm1.Close
EndSub

Sub txt1_Change ()
  If txt1.Text = "" Then    ' See if text box is
    empty.
    cmd1.Enabled = False    ' Disable button.
  Else
    cmd1.Enabled = True    ' Enable button.
  End If
End Sub
```

See Also

Form, Ocx

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

Flags Property (CommDlg)

Purpose

Returns or sets the options for a common dialog box.

Syntax

CommDlg.**Flags** [= *long*]

Description

The **Flags** property specifies the options for a common dialog box. The **Flags** property is shared by all common dialog boxes. Each dialog box has its own set of predefined flags. These flag values are listed in the **Show** methods of the **CommDlg** object.

ShowOpen	Show Open Dialog Box
ShowSave	Show Save As Dialog Box
ShowColor	Show Color Dialog Box
ShowFont	Show Font Dialog Box
ShowPageSetup	Show Page Setup Dialog Box
ShowPrint	Show Print or Print Options Dialog Box
ShowHelp	Invokes the Windows Help Engine for .hlp files only; see Accessing HTMLHelp Files for how to access .chm help files.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

ShowFont Method, FontName, FontItalic, FontBold, FontUnderline, FontStrikethru, FontSize, Font, PointSize, FontStyle, Min, Max Properties

Purpose

Displays the **CommDlg** control's Font dialog box.

Syntax

CommDlg.**ShowFont**

CommDlg.**FontName** [= *string*]
CommDlg.**FontItalic** [= *Bool*]
CommDlg.**FontBold** [= *Bool*]
CommDlg.**FontUnderline** [= *Bool*]
CommDlg.**FontStrikethru** [= *Bool*]
CommDlg.**FontSize** [= *single*]
CommDlg.**Font** [= *Font*]

CommDlg.**PointSize** [= *integer*]
CommDlg.**FontStyle** [= *string*]
CommDlg.**Min** [= *integer*]
CommDlg.**Max** [= *integer*]

Description

Before you use the **ShowFont** method, you must set the **Flags** property of the **CommDlg** object to one of three constants or values: **cdfBoth** (3), **cdfPrinterFonts** (2), or **cdfScreenFonts** (1).

The **Flags** property returns or sets the options for the Font dialog box and can have one or more of the following values.

cdfAnsiOnly $400 - Only fonts that use the Windows character set (no symbol font).

cdfApply $200 - Enables the Apply button on the dialog box.

cdfBoth $3 - Both printer and screen fonts. The **hDC** property identifies the device context associated with the printer.

cdfEffects $100 - Enables strikethrough, underline, and color effects.

cdfFixedPitch $400 - 0Only fixed-pitch fonts.

cdfForceFontExists $10000 - An error message box is displayed if the user attempts to select a font or style that doesn't exist.

cdfInitFont $40 - Use GFA-BASIC 32 internal **LOGFONT** structure to initialize the dialog box controls.

cdfShowHelp $4 - Causes the dialog box to display a Help button.

cdfLimitSize $2000 - Only font with sizes within the range specified by the **Min** and **Max** properties.

cdfNoOEMFonts $800 - Don't allow OEM font selections

cdfNoScriptSel $800000 - Disables the **Script** combo box (only used to initialize the dialog box).

cdfNoFaceSel $80000 - No font name selected.

cdfNoSimulations $1000 - Don't allow graphic device interface (GDI) font simulations.

cdfNoSizeSel $200000 - No font size selected.

cdfNoStyleSel $100000 - No style was selected.

cdfNoVector $800 - Don't allow vector-font selections.

cdfNoVertFonts $1000000 - Don't allow vertical fonts selections.

cdfPrinterFonts $2 - Only the fonts supported by the printer, specified by the **hDC** property.

cdfScalableOnly $20000 - Only fonts that can be scaled.

cdfScriptsOnly $400 - Allow selection of fonts for all non-OEM and Symbol character sets, as well as the ANSI character set. This supersedes the **cdfAnsiOnly** value.

cdfScreenFonts $1 - Only the screen fonts supported by the system.

cdfTTOnly $40000 - Only TrueType fonts.

cdfSelectScript $400000 - Only fonts with the character set identified in the **lfCharSet** member of the internal **LOGFONT** structure are displayed.

cdfWysiwyg $8000 - Only fonts that are available on both the printer and on screen. If this flag is set, the **cdfBoth** and **cdfScalableOnly** flags should also be set.

cdfUseStyle $80 - Use the data in the **FontStyle** property to initialize the font style combo box. When the dialog is closed the combobox data is copied to **FontStyle**.

In general, you should change **FontName** before setting size and style attributes with the **FontSize**, **FontBold**, **FontItalic**, **FontStrikethru**, and **FontUnderline** properties. For detailed information: See Also.

The **FontStyle** [= *string*] property contains the style data ("Bold", "Normal") for the Style combobox of the dialog. The string is a regional setting name.

The **PointSize** [= *integer*] property specifies the size of the selected font, in units of 1/10 of a point.

The **Min** property specifies the minimum point size a user can select. The **Max** property specifies the maximum point size a user can select. **ShowFont** recognizes this member only if the **cdfLimitSize** flag is specified.

Example

```
Print
Ocx CommDlg cd
cd.Flags = cdfScreenFonts | cdfUseStyle |
 cdfEffects
cd.CancelError = True
Try
  cd.ShowFont
  ForeColor = cd.Color
  Set Me.Font = cd.Font // select font
  Print "1234", Me.FontName
```

```
Catch
  Print "Common dialog box canceled!"
EndCatch
Do
  Sleep
Until Me Is Nothing
```

See Also

CommDlg, Dlg Font, Font, FontSize, FontBold, FontItalic, FontStrikethru, FontUnderline

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

ShowHelp Method, HelpFile, HelpContext, HelpKey, HelpCommand Property

Purpose

The **CommDlg** method **ShowHelp** invokes WinHelp and displays the .hlp Help file you specify; for .chm Help files, see Accessing HTMLHelp files.

Syntax

CommDlg.**ShowHelp**

*CommDlg*_**HelpFile** [= *string*]
*CommDlg*_**HelpContext** [= *integer*]
*CommDlg*_**HelpKey** [= string]
*CommDlg*_**HelpCommand** [= *integer*]

Description

The **ShowHelp** method calls Winhlp32.exe for the help file specified in the **HelpFile** property and in the mode specified **HelpCommand**.

The **HelpFile** property specifies the path and filename of the Help file to display with **ShowHelp**.

HelpContext returns or sets the context ID of the requested Help topic.

HelpKey returns or sets the keyword that identifies the requested Help topic.

The **HelpCommand** property returns or sets the type of online Help requested. This value should be one of the following constants.

cdhCommand (258)	Executes a Help macro.
cdhContents (3)	Displays the Help contents topic as defined by the Contents option in the [OPTION] section of the .hpj file. See Remarks below for information on Help files created with Microsoft Help Workshop 4.0X.
cdhContext (1)	Displays Help for a particular context. When using this setting, you must also specify a context using the **HelpContext** property.
CdhContextPopup (8)	Displays in a pop-up window a particular Help topic identified by a context number defined in the [MAP] section of the .hpj file.
cdhFinder (11)	Displays the **Help Topics** dialog box.
cdhForceFile (9)	Ensures WinHelp displays the correct Help file. If the correct Help file is currently displayed, no action occurs. If the incorrect Help file is displayed, WinHelp opens the correct file.
cdhHelpOnHelp (4)	Displays Help for using the Help application itself.
cdhIndex (3)	Displays the index of the specified Help file. An application should use

	this value only for a Help file with a single index.
cdhKey (257)	Displays Help for a particular keyword. When using this setting, you must also specify a keyword using the **HelpKey** property.
cdhMultiKey (513)	Displays the topic specified by a keyword in an alternative keyword table. **HelpContext** must contain the ASCII code of a single character that identifies the keyword table to search and **HelpKey** should specify the text string that specifies the keyword to locate in the keyword table.
cdhPartialKey (261)	Displays the topic found in the keyword list that matches the keyword passed in the **HelpKey** property if there is one exact match.
cdhQuit (2)	Notifies the Help application that the specified Help file is no longer in use.
cdhSetContents (5)	Determines which contents topic is displayed when a user presses the F1 key.
cdhPopupPos (13)	Sets the context specified by the **HelpContext** property as the current index for the Help file specified by the **HelpFile** property. This index remains current until the user accesses a different Help file. Use this value only for Help files with more than one index.

Example

```
Print
Local d$ = Left(ProgName$, RInStr(ProgName$, "\"))
  & "GfaWin32.hlp"
Ocx CommDlg cd
cd.HelpFile = d$
cd.HelpKey = "Form"
cd.HelpCommand = cdhKey
cd.ShowHelp
CloseW 1
```

See Also

CommDlg

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Left, Top Properties

Purpose

Return or set the position of an Ocx object.

Syntax

object.**Left** [= value]

object.**Top** [= value]

object:Ocx objects
value:Single exp

Description

The **Left** and **Top** properties set the position of an OCX control or Form. The *value* is specified in pixels. For OCX controls, the units can be adjusted to the current scaling of the parent Form. The Form property **OcxScale** = True sets the coordinate scheme for the Ocx controls to the **ScaleMode** of the Form. By default the **ScaleMode** = **basPixels** (in VB mostly twips).

Example

```
Form Frm
Print "Click to centre the form"
Do
  Sleep
Until Me Is Nothing

Sub Frm_Click ()
```

```
  With Frm
    .Width = Screen.Width * .75             ' Set
     width of form.
    .Height = Screen.Height * .75           ' Set
     height of form.
    .Left = (Screen.Width - .Width) / 2   ' Center
     form horizontally.
    .Top = (Screen.Height - .Height) / 2 ' Center
     form vertically.
  End With
  Cls
  Print "Now close the form"
End Sub
```

This example sets the size of a form to 75 percent of screen size and centers the form when it is loaded.

See Also

Form, Left, Top, Move, OcxScale, ScaleMode

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

Parent Property

Purpose

Returns the parent Form object for the given OCX.

Syntax

Set f = object.**Parent**

object:Ocx Object
f:Form Object

Description

Parent is used to get the parent window for an Ocx. Use the **Parent** property to access the properties, methods, or controls of an object's parent.

Example

```
OpenW 1
Ocx Command cmd1 = "Move Parent", 10, 10, 100, 40
Do
  Sleep
Until Me Is Nothing

Sub cmd1_Click
  MoveParent(cmd1)
EndSub

Sub MoveParent(o As Object)
  ' Move Parent to a random position
  If TypeOf(o.parent) Is Form
```

```
    o.Parent.Move PixelsToTwipX(Random(300)),
      PixelsToTwipY(Random(200))
  EndIf
EndSub
```

Remarks

The **Parent** property is useful in an application in which you pass objects as arguments. For example, you could pass a control variable to a general procedure, and use the **Parent** property to access its parent form.

There is no relationship between the **Parent** property and the **MdiChild** property. There is, however, a parent-child relationship between an **MdiParent** object and any **Form** object that has its **MdiChild** property set to **True**.

See Also

Form, MdiParent

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

ShowPageSetup Method, pgBottom, pgLeft, pgRight, pgTop, pgMinBottom, pgMinLeft, pgMinRight, pgMinTop, pgScale

Purpose

Creates a **Page Setup** dialog box that enables the user to specify the attributes of a printed page. These attributes include the paper size and source, the page orientation (portrait or landscape), and the width of the page margins.

Syntax

CommDlg.**ShowPageSetup**

CommDlg.**pgBottom** [= *long*]
CommDlg.**pgLeft** [= *long*]
CommDlg.**pgRight** [= *long*]
CommDlg.**pgTop** [= *long*]
CommDlg.**pgMinBottom** [= *long*]
CommDlg.**pgMinLeft** [= *long*]
CommDlg.**pgMinRight** [= *long*]
CommDlg.**pgMinTop** [= *long*]
CommDlg.**pgScale** [= *long*]

CommDlg.**DevNames** [= *string*]

Description

The **ShowPageSetup** method enables to set printer page attributes, including the paper size and source, the page orientation (portrait or landscape), and the width of the page margins.

The **pgBottom**, **pgLeft**, **pgRight**, and **pgTop** properties return or set the widths of the left, top, right, and bottom margins for your document. The **Flags** property must include the **cdpsMargins.** The margin units are determined by **pgScale**. **pgScale** = 1 indicates that hundredths of millimeters (1/100 mm) are the unit of measurement for margins and paper size. When **pgScale** = 2 thousandths of inches (1/1000 inch) is the measurement unit. Setting the property implicitly modifies **Flags** with either $4 or $8.

The **pgMinBottom**, **pgMinLeft**, **pgMinRight**, and **pgMinTop** properties set the minimum allowable values for the left, top, right, and bottom margin input boxes of the dialog box. Input below these values is reset to the minimum settings specified. The **Flags** property must include the **cdpsMinMargins.**

The DevNames property returns the selected printer as a string with, comma delimited, Driver, Device, and Output Port. For example "WINSPOOL,HP Laserjet 4,LPT1:".

The **Flags** property values for **ShowPageSetup** are:

Flags	**Meaning**
cdpsMinMargins $1	The **pgMinBottom**, **pgMinLeft**, **pgMinRight**, and **pgMinTop** properties are used to initialize the dialog box.
cdpsMargins $2	The **pgBottom**, **pgLeft**, **pgRight**, and **pgTop**

	properties are used to initialize the dialog box.
$4	Hundredths of millimeters are the unit of measurement for margins and paper size (set by **pgScale** = 1).
$8	Thousandths of inches are the unit of measurement for margins and paper size (set by **pgScale** = 2).
cdpsDisableMargins $10	Disables the margin controls, preventing the user from setting the margins.
cdpsDisablePrinter $20	Disables the **Printer** button, preventing the user from invoking a dialog box that contains additional printer setup information.
cdpsNoWarning $80	Prevents the system from displaying a warning message when there is no default printer.
cdpsDisableOrientation $100	Disables the orientation controls, preventing the user from setting the page orientation.
cdpsReturnDefault $400	Returns the standard printer in **DevNames** without showing the dialog box.
cdpsDisablePaper $200	Disables the paper controls, preventing the user from setting page parameters

	such as the paper size and source.
cdpsShowHelp $800	Displays the **Help** button
cdpsDisablePagePainting $80000	Prevents the dialog box from drawing the contents of the sample page.
cdpsNoNetworkButton $2000000	Hides and disables the **Network** button.

Example

```
Ocx CommDlg cd
cd.Flags = cdpsMargins | cdpsMinMargins
cd.pgScale = 1              ' 1/100 mm
cd.pgBottom = 1000          ' 10 mm
cd.pgLeft = 1000            ' 10 mm
cd.pgRight = 1000           ' 10 mm
cd.pgTop = 1000             ' 10 mm
cd.pgMinBottom = cd.pgBottom
cd.pgMinLeft = cd.pgLeft
cd.pgMinRight = cd.pgRight
cd.pgMinTop = cd.pgTop
cd.ShowPageSetup
Debug.Show
Trace cd.pgBottom       ' use for the documents
Trace cd.pgLeft
Trace cd.pgRight
Trace cd.pgTop
Trace cd.DevNames       ' The selected device
Set Printer = cd        ' Assign to Printer object
' Show the new Printer settings for the device …
Trace Printer.DefLeft
Trace Printer.Width
Trace Printer.DeviceName
Trace Printer.Orientation
Trace Printer.dmPaperSize
```

```
Me.Close
```

Remarks

The return values in **pgBottom**, **pgLeft**, **pgRight**, and **pgTop** return margin settings for your documents. They have no relation whatsoever with the capabilities of the printer. Other settings made with **ShowPageSetup** common dialog box are available only when you make the selection the default for the application. This accomplished by setting the **CommDlg** object as the new **Printer** object. The **Printer** object is then initialized using the DEVMODE structure which is a shared object between the Print and PageSetup dialog box. The DEVMODE fields set with the **ShowPageSetup** dialog box are then available through the **Printer's** (device mode dm) properties.

When the **ShowPrint** dialog box is displayed it uses the shared DEVMODE structure, which might have been changed through the use of the **Printer** properties, to fill in the controls of the dialog box. Changing printing attributes in the **ShowPrint** dialog box doesn't make the effective until you re-assign the **CommDlg** object to the **Printer** object.

See Also

CommDlg, ShowPrint, Dlg Print, dm-Properties

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Tag Property

Purpose

Returns or sets an expression that stores any extra data (string) needed for your program. Unlike other properties, the value of the **Tag** property isn't used by GFA-BASIC 32; you can use this property to identify objects.

Syntax

object.**Tag** [= *exp*]

object:Ocx object
exp:String expression

Description

You can use this property to assign an identification string to an object without affecting any of its other property settings or causing side effects. The **Tag** property is useful when you need to check the identity of a control or MDI Form object that is passed as a variable to a procedure.

Example

```
Ocx Command cmd = "This is the Caption Text", 10,
 10, 140, 22 : cmd.Tag = "This is the Tag Text"
Ocx CheckBox chk = "Show Tag Text in Caption", 10,
 40, 160, 14
Do : Sleep : Until Me Is Nothing

Sub chk_Click
 Select chk.Value
```

```
  Case 0 : cmd.Caption = "This is the Caption Text"
  Case 1 : cmd.Caption = cmd.Tag
  EndSelect
EndSub
```

Remarks

As an alternative, the HelpContextID property can be used as a place to store additional data. This property is an Integer and therefore it's performance much better.

See Also

Form

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

AboutBox Method (Ocx controls)

Purpose

Displays the About box for the control.

Syntax

object.**AboutBox**

object:Ocx object

Description

Only some of the Ocx controls support an **AboutBox**. According to MS documentation all ActiveX controls should support an **AboutBox**.

Example

```
Ocx MonthView mvw = "", 10, 10, 0, 0
mvw.AboutBox
```

See Also

MonthView

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

ShowFolders Method

Purpose

Creates a dialog box that allows the user to select a folder.

Syntax

CommDlg.**FileName** [= *string*]

CommDlg.**ShowFolders**[(*flag*)]

CommDlg.**Title** [= *string*]

Description

Displays a dialog box that enables the user to select a shell folder. The optional *flag* (Variant) specifies the style for the dialog box.

The **Title** property can be used to set a customized dialog box title, while the **FileName** property serves both as a receptable for the default folder beforehand and the file path once a folder has been chosen.

Example

```
Public Const BIF_RETURNONLYFSDIRS = $1
Public Const BIF_DONTGOBELOWDOMAIN = $2
Public Const BIF_STATUSTEXT = $4
Public Const BIF_RETURNFSANCESTORS = $8
Public Const BIF_EDITBOX  =  0x0010
Public Const BIF_VALIDATE =  0x0020   // insist on
 valid result (or CANCEL)
```

```
Public Const BIF_NEWDIALOGSTYLE = 0x0040   // Use
  the new dialog layout with the ability to resize.
Public Const BIF_USENEWUI = (BIF_NEWDIALOGSTYLE |
  BIF_EDITBOX)
Public Const BIF_NONEWFOLDERBUTTON = 0x0200   //
  Do not add the "New Folder" button to the
  dialog.  Only applicable with
  BIF_NEWDIALOGSTYLE."
Public Const BIF_BROWSEFORCOMPUTER = $1000
Public Const BIF_BROWSEFORPRINTER = $2000
Public Const BIF_BROWSEINCLUDEFILES = 0x4000  //
  Browsing for Everything
Ocx CommDlg cd
cd.Title = "Select directory"
cd.FileName = "F:\James Documents\My Games"   //
  Use cd.Filename to set the default Folder
cd.ShowFolders BIF_USENEWUI
Print cd.FileName
```

Remarks

To be used in GB32, the folder path returned by *cd*.**FileName** needs to be converted using **ZTrim**, as is shown by the example below:

```
Ocx CommDlg cd
cd.FileName = "c:\windows"
cd.ShowFolders
If cd.FileName <> ""
  Print cd.FileName & "\"
  Print ZTrim(cd.FileName) & "\"
EndIf
```

Known Issues

Noted recently is what could be described as the 'renamed folder' error: if you rename a folder in the Commdlg

ShowFolders window and click 'OK' before finishing the edit (before pressing Enter or clicking on another folder), the folder will be renamed, but the value returned in *cd*.**FileName** will be that of the name of the folder before it was renamed, which will cause an error if you then try to access it; this happens even if you use the BIF_VALIDATE flag. This is not truly a GFA bug, but something to be aware of.
[Reported by James Gaite, 01/03/2017]

This Ocx object has had a very on/off performance with it working well sometimes and not at all at other times. Tested on Win8.1 using GFA IDE build 1169 with OCX build 1185, the above example works - this has not always been the case.

If you run into problems with **ShowFolders**, there are two available workarounds:

```
Function COMBrowseForFolder(Flags As Long) As
  String
  // Courtesy of Sjouke Hamstra
  Local Object oShell, oFolder, oFolderItem
  Const ssfDRIVES = &H11       ' from
    ShellSpecialFolderConstants
  Try
    ' Create a shell object like it is done in
      VBScript
    Set oShell   =
      CreateObject("Shell.Application")
    ' BrowseForFolder returns an object of the
      Folder data type
    ' The Shell object model's Folder object is the
      COM representation of a Windows folder.
    ' The Folder object contains a collection of
      child objects, each representing an
    ' item in the folder. Hence, these child
      objects are called FolderItem objects.
    Set oFolder  = oShell.BrowseForFolder(Null, _
      "Select or type the folder where you want to
        begin the search.", Flags, ssfDRIVES)
    ' oFolder.Title is the default property and
      returns a
    ' a string that is exactly the text you
      highlighted
    Trace oFolder.Title  'the title of the folder.
    If (Not oFolder Is Nothing) Then
      ' Transform the folder into a FolderItem
        object
      Set oFolderItem = oFolder.Items.Item
    EndIf
    COMBrowseForFolder = oFolderItem.path
    ' Trace oFolderItem.path
  Catch
  EndCatch
  Set oFolderItem = Nothing
  Set oFolder = Nothing
  Set oShell = Nothing
EndFunc
```

...or...

```
Declare Function SHGetPathFromIDList Lib
  "shell32.dll" (ByVal pidl As Long, _
  ByVal pszBuffer As String) As Long
Declare Function SHBrowseForFolder Lib
  "shell32.dll" (lpBrowseInfo As _
  BROWSEINFO) As Long
Type BROWSEINFO
  hOwner              As Long
  pidlRoot            As Long
  pszDisplayName   As Long
  lpszTitle        As Long
  ulFlags          As Long
  lpfn             As Long
  lParam           As Long
  iImage           As Long
EndType
Const BIF_RETURNONLYFSDIRS As Long = &H1
Const BIF_DONTGOBELOWDOMAIN As Long = &H2
Const BIF_RETURNFSANCESTORS As Long = &H8
Const BIF_EDITBOX  =  &H10
Const BIF_VALIDATE =  &H20
Const BIF_NEWDIALOGSTYLE = &H40
Const BIF_USENEWUI = (BIF_NEWDIALOGSTYLE |
  BIF_EDITBOX)
Const BIF_NONEWFOLDERBUTTON = &H200
Const BIF_BROWSEFORCOMPUTER As Long = &H1000
Const BIF_BROWSEFORPRINTER As Long = &H2000
Const BIF_BROWSEINCLUDEFILES As Long = &H4000
Const BFFM_SETSELECTION = (WM_USER + 102)
Dim foldir$ = App.Path & "\" : Print foldir$
OpenW 1
Print BrowseForFolder(Win_1.hWnd, "Title") : Print
  foldir$
Print BrowseForFolder(Win_1.hWnd, "Title") : Print
  foldir$

Function BrowseForFolder(hnd%, Title As String,
  Optional Flags%) As String
  Local bi As BROWSEINFO
  Local Int pidl
  Local path$ = Space$(512) + #0, buf$ =
    Space$(512) + #0
  If Flags <= 0 Then Flags = BIF_RETURNONLYFSDIRS |
    BIF_USENEWUI
  Title = Title + #0
  bi.hOwner = hnd
  bi.pidlRoot = 0
  bi.lpszTitle = V:Title
  buf = Space$(512) + #0
  bi.pszDisplayName = V:buf
  bi.ulFlags = Flags
  bi.lpfn = ProcAddr(BrowseCallbackProc)
  pidl = SHBrowseForFolder(bi)
  If pidl
    If SHGetPathFromIDList(pidl, path)
      path = ZTrim$(path)
      If Right$(path, 1) <> "\" Then path = path +
        "\"
      foldir$ = path$
    Else
      path = "Error"
    EndIf
  Else
    path = ""
  EndIf
  BrowseForFolder = path
  ~CoTaskMemFree(pidl)
EndFunc

Function BrowseCallbackProc(hwnd As Handle, uMsg
  As Int, lp%, pData%) As Int
  Switch(uMsg)
  Case 1                    'BFFM_INITIALIZED :
    ~SendMessage(hwnd, BFFM_SETSELECTION, True,
      V:foldir$)
  Case 2
    Print "This is option 2"
  EndSelect
  Return 0
EndFunction
```

See Also

CommDlg

{Created by Sjouke Hamstra; Last updated: 01/03/2017 by James Gaite}

Dlg Open, Dlg Save Command

Purpose

Calls the common file selecting dialog box.

Syntax

Dlg Open form, Flags%, Title$, Dir$, DefExt$, Filter$[()], Ret$

Dlg Save form, Flags%, Title$, Dir$, DefExt$, Filter$[()], Ret$

Description

Dlg Open and **Dlg Save** call, like the command **FileSelect**, the common file selecting dialog. However, in contrast with **FileSelect**, **Dlg Open** and **Dlg Save** can be configured.

form is a Form object, like **Me**, **Win_1**, **Dlg_1**, frm1

Title$ is the title of the file dialog box.

Dir$ is the default directory (a string).

DefExt$ is a file name extension of three characters, which will automatically be appended if no extension is given.

Filter$* is a either a string array or a string declaring the file search filter. Two strings apply to each selection: the first contains descriptive text which appears in the ComboBox. The next one determines the file mask (filter) which applies to it. (It can hold multiple examples each separated by a semicolon ";"). When using an array terminate the array with an empty string, see example. *Filter$ may also be a string. Use the pipe (|) symbol (ASCII 124) to separate the *description* and *filter* values. Don't include spaces before or after the pipe symbol, because these spaces will be displayed with the *description* and *filter* values. For instance:

flit$ = "Text (*.txt)|*.txt|Pictures (*.bmp;*.ico)|*.bmp;*.ico"

Ret$ is a string variable which receives the file name. The string will contain the full path. When the dialog box is canceled the return value is an empty string.

Flags can contain the following values:

OFN_READONLY	$00001	The Read-Only check box will be activated. Return value in _EBX.
OFN_OVERWRITEPROMPT	$00002	If a file already exists a warning appears.
OFN_HIDEREADONLY	$00004	Hide the Read-Only checkbox.
OFN_NOCHANGEDIR	$00008	Resets back to the same directory as when the Dialog was created.
OFN_NOVALIDATE	$00100	Invalid characters in the filename are allowed
OFN_EXTENSINODIFFERENT	$00400	Returns a value in

		_EBX if extension is different than specified.
OFN_PATHMUSTEXIST	$00800	Selected path is verified.
OFN_FILEMUSTEXIST	$01000	Selected file is verified.
OFN_CREATEPROMPT	$02000	If a file does not exist, a warning is displayed.
OFN_NOREADONLYRETURN	$08000	Does not return names of write-protected files.
OFN_NOTESTFILECREATE	$10000	With Open SAVE, the file will not be created and cancelled for the purposes of testing. This option was conceived for WORM (WriteOnce Read Many) drives, create-no-modify networks, and the like.

These bits are not allowed in GFA-BASIC:

OFN_SHOWHELP	$00010
OFN_ENABLEHOOK	$00020
OFN_ENABLETEMPLATE	$00040
OFN_ENABLETEMPLATEHANDLE	$00080
OFN_ALLOWMULTISELECT	$00200

OFN_SHAREAWARE $04000

_AX is a null if there is an error.

File$ is selected filename and path.

_EBX is the new value for Flags.

Example

```
OpenW 1
Auto file$
Dim filt$(20)
filt$(0) = "BMP-Files", filt$(1) = "*.BMP;*.RLE"
filt$(2) = "PCX-Files", filt$(3) = "*.PCX"
filt$(4) = ""
file$ = "NONAME.BMP"
Dlg Open Me, 0, "This is a test", "d:\pcx", "BMP",
  filt$(), file$
Dlg Open Me, 0, "", "", "BMP", "BMP-
  Files|*.BMP;*.RLE|PCX files|*.PCX", file$
```

This code fragment specifies two filters. The filter with the "BMP-Files" description has two patterns. If the user selects this filter, the dialog box displays only files that have the .BMP and .RLE extensions.

Remarks

This command is implemented for compatibility reasons only. Use **CommDlg** object instead.

See Also

CommDlg, Dlg Font, Dlg Color, Dlg Print

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

Dlg Color Command

Purpose

calls the common color selecting dialog box for a form.

Syntax

Dlg Color form, Flags , custcol(), color

formForm object
Flagsiexp (CC_ constants)
cust()ivar
color:ivar

Description

form - A Form object, for instance **Me**, Win_1, Dlg_1, frm1, etc.

Flags - Sets several options of the Dialog box.

CC_RGBINIT ($1)	*color* will be used as default.
CC_FULLOPEN ($2)	The whole Dialog box appears immediately, otherwise just the left side with pre-defined Custom Colors appears.
CC_PREVENTFULLOPEN($4)	The right side of the Dialog box is switched off, preventing the creation of new Custom Colors.
Not allowed are:	CC_SHOWHELP ($8) CC_ENABLEHOOK ($16) CC_ENABLETEMPLATE ($32)

CC_ENABLETEMPLATEHANDLE ($64)

custcol() - A long integer field, holds the Custom Colors in first 16 elements.

color - A long integer variable, holds a default color-selection. In this command, colors are always RGB values.

_AX is a null if there is an error. Otherwise, color% is the newly selected color. cust%() is filled with the new Custom Colors.

Example

```
Print                              // Open window #1
Dim col(0 .. 15) As Int, c As Int, i As Int
For i = 0 To 15
  col(i) = QBColor(i)             // Custom colors
Next
Dlg Color Me, 0, col(), c          // Dialog
If _AX
  Color c : Print (Hex(c))        // Print in color
  For i = 0 To 15
    Color col(i) : Print Hex(col(i), 8),
    If i Mod 4 = 0 Then Print
  Next
  Color 0 : Print (Hex(c))        // Print in black
  For i = 0 To 15
    Print Hex(col(i), 8),
    If i Mod 4 = 0 Then Print
  Next
EndIf
```

This small program produces a color-selector and evaluates the selection.

Remarks

This command is implemented for compatibility reasons only. Use **CommDlg** object instead.

See Also

CommDlg, Color, Dlg Font, Dlg Open, Dlg Print, Rgb(), RGBColor

{Created by Sjouke Hamstra; Last updated: 02/10/2014 by James Gaite}

Dlg Print Command

Purpose

calls the common printer selecting dialog box. Implemented for compatibility reasons only. It is advised not to use this command and instead use CommDlg object.

Syntax

Dlg Print form, Flags%, hDC

Description

form is a Form object like me, Win_1, Dlg_1, frm1.

Flags% declares some bit-wise settings:

PD_ALLPAGES	$000000	Sets all Radio Buttons On.
PD_SELECTION	$000001	Sets the selection Radio Button to On.
PD_PAGENUMS	$000002	Sets the pages Radio Button to On.
PD_NOSELECTION	$000004	Print selection disabled.
PD_NOPAGENUMS	$000008	Page numbers disabled.
PD_COLLATE	$000010	Sets the Collate Copies check box to On.
PD_PRINTTOFILE	$000020	Sets the Print to File check box to

		On.
PD_PRINTSETUP	$000040	Calls the Setup Dialog. (The Setup Button also allows the user to call up the Setup Dialog directly.)
PD_NOWARNING	$000080	Warnings about errors in the Default Printer are suppressed.
PD_USEDEVMODECOPIES	$040000	If a printer driver can make copies itself it is used instead of the Print Manager.
PD_DISABLEPRINTTOFILE	$080000	Print to File is disabled.
PD_HIDEPRINTTOFILE	$100000	The Print to File check box is hidden.

GFA-BASIC does not allow the following:

PD_RETURNDC	$000100 always returns a DC.
PD_RETURNIC	$000200
PD_RETURNDEFAULT	$000400
PD_SHOWHELP	$000800
PD_ENABLEPRINTHOOK	$001000
PD_ENABLESETUPHOOK	$002000
PD_ENABLEPRINTTEMPLATE	$004000
PD_ENABLESETUPTEMPLATE	$008000

PD_ENABLEPRINTTEMPLATEHANDLE	$010000
PD_ENABLESETUPTEMPLATEHANDLE	$020000

hDC is the return value, this is a device context like **PrinterDC**().

_AX is null, if an error has occurred. Otherwise, _BX is the "from page(i.e.: Starting Page).

_CX is the "to" page(i.e.: Ending Page).

_EX is the number of copies.

_EFL holds the new values for the flags:

_EFL %& PD_PAGEENUMS is not equal to zero when a range of page number button is chosen

_EFL %& PD_SELECTION is not equal to zero when selection button is chosen, and so forth.

_SI is the handle of the internal hDevMode structure.

_DI is the handle of the internal hDevNames structure.

Special: When **_AX** = $1234 some parameters can be set prior to the call from **Dlg Print**.

_BX	beginning page;
_CX	ending page,
_DX	number of copies;
_SI	smallest page number;
_DI	largest page number (without these settings, the page numbers lie between 0 and 100).

Example

```
OpenW 1
Local h As Handle
Dlg Print Win_1, 0, h
If Not IsNull(h)
  SetPrinterHDC h
  Output = Printer
  Printer.FontSize = 12 : Printer.FontName =
    "Arial"
  Printer.StartDoc "GFA Test"
  Printer.StartPage
  Print "Hello World"
  Printer.FontName = "courier new"
  Text 200, 400, "Hello World 2"
  Printer.EndPage
  Printer.EndDoc
  Output = Win_1
EndIf
CloseW 1
```

Alternatively, you can use CommDlg Print as in the following example:"

```
OpenW 1
Print " Start printing"
//
// to use the Ocx Commdlg with the name cd
Ocx CommDlg cd
// to make Cancel possible
cd.CancelError = True
// to show the printer dialog
cd.ShowPrint
//to choose the printer and to activate it in the
  following
Set Printer = cd
// to set one flag
cd.Flags = cdpDisablePrintToFile
// all output to the printer
```

```
Output = Printer
// to start the print job
Printer.StartDoc "Text"
// 1. start page of your printing
Printer.StartPage
// to use a font
Printer.FontName = "Arial"
// to use a font size
Printer.FontSize = 16
Print "Hello GFA"
DefLine 10, 2
Circle 100, 100, 300
Box 150, 150, 240, 240
// end ot the page
Printer.EndPage
// end of the print job
Printer.EndDoc
// output back into the actual window
Output = Win_1
Print "printing is finished"
Print
Print "press Alt F4 to end"
Do : Sleep : Until Me Is Nothing
```

See Also

CommDlg, Printer

{Created by Sjouke Hamstra; Last updated: 24/11/2015 by James Gaite}

CurrentX, CurrentY Properties

Purpose

Returns or sets the horizontal (**CurrentX**) or vertical (**CurrentY**) coordinates for the next drawing method.

Syntax

[Object.]**CurrentX** [= value]

[Object.]**CurrentY** [= value]

Object:Form or Printer object
value: Single expression

Description

Coordinates are measured from the upper-left corner of an object. The **CurrentX** property setting is 0 at an object's left edge, and the **CurrentY** property setting is 0 at its top edge. Coordinates are expressed in pixels, or the current unit of measurement defined by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, **ScaleTop**, and **ScaleMode** properties.

When you use the following graphics methods, the **CurrentX** and **CurrentY** settings are changed as indicated:

[**P**]**Circle**	The center of the object.
[**P**]**Box**	The right bottom corner of the object.
[**Gray**]**Text**	The right bottom corner of the text.

Cls	0, 0.
Draw/Line	The end point of the line.
Point/Pset	The point drawn.
EndDoc	0, 0
NewPage	0, 0
[**L**]**Print**	The next print position

Example

```
OpenW 1
Local a%, xx, yy
Win_1.CurrentX = 150
Win_1.CurrentY = 50
xx = CurrentX
yy = CurrentY
Text CurrentX, CurrentY, "Press any key"
KeyGet a%
Cls
Text xx, yy, "GFA"
Circle 100, 100, 50
Text CurrentX, CurrentY, "X"
Do : Sleep : Until Me Is Nothing
```

Remarks

Usually, **CurrentX** and **CurrentY** are use with an object, like **Me.CurrentX**, or **Printer.CurrentX**. Without an object, the current **Output** device is used.

See Also

Output, Form, Printer

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

DefHeight, DefLeft, DefTop, DefWidth, Height, Left, Top, Width Properties (Printer)

Purpose

The Defxx properties return the printer's default left, top, height and width settings.

Syntax

Printer.**DefHeight**
Printer.**DefLeft**
Printer.**DefTop**
Printer.**DefWidth**

Printer.**Height** [= *single*]
Printer.**Left** [= *single*]
Printer.**Top** [= *single*]
Printer.**Width** [= *single*]

Description

The printable area of a page is returned in the **DefHeight**, **DefLeft**, **DefTop**, and **DefWidth** properties. The return value is of type Single.

By default, the **DefHeight**, **DefLeft**, **DefTop**, and **DefWidth** properties are identical to the **Height**, **Left**, **Top**, and **Width** properties. However the **Height**, **Left**, **Top**, and **Width** properties can be used to set the physical dimensions of the paper.

The coordinates are in **ScaleMode** units.

Example

```
Debug.Show
Trace Printer.ScaleMode
Trace Printer.DefLeft
Trace Printer.DefTop
Trace Printer.DefHeight
Trace Printer.DefWidth
Trace Printer.Left
Trace Printer.Top
Trace Printer.Height
Trace Printer.Width
```

See Also

Printer, ShowPrint, ShowPageSetup, Height, Left, Top, Width

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

DriverName, DeviceName, PortName Properties (Printer)

Purpose

Returns the name of the driver for a **Printer** object, the name of the device a driver supports, and the name of the port through which a document is sent to a printer.

Syntax

Printer.**DeviceName**
Printer.**DriverName**
Printer.**PortName**

Description

Each driver has a unique name. For example, the **DriverName** for several of the Hewlett-Packard printers is HPPCL5MS. The **DriverName** is typically the driver's filename without an extension.

The **DeviceName** property contains the name of the device the driver supports. For Example, "PCL/HP LaserJet" is the name of one driver. You can use this to indicate the printer you're printing on.

The **PortName** returns the name of the port which is determined by the operating system determines, such as LPT1: or LPT2:.

Example

```
Ocx CommDlg cd
cd.Flags = 0
cd.ShowPageSetup
Set Printer = cd
Debug.Show
Trace cd.DevNames
Trace Printer.DeviceName
Trace Printer.DriverName
Trace Printer.PortName
```

Remarks

The properties of the **Printer** object initially match those of the default printer set in the Windows Control Panel.

See Also

Printer, ShowPageSetup, ShowPrint, SetPrinterByName

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

dmCollate, dmColor, dmCopies, dmPaperBin, dmPaperBinName, dmPaperLength, dmPaperSize, dmPaperSizeName, dmPaperSizeX, dmPaperSizeY, dmPaperWidth, dmQuality, dmYRes Properties

Purpose

Return or set device mode properties of the **Printer** object.

Syntax

Printer.**dmCollate** [= *long*]
Printer.**dmColor** [= *long*]
Printer.**dmCopies** [= *long*]
Printer.**dmPaperBin** [= *string*]
Printer.**dmPaperBinName(**i%**)**
Printer.**dmPaperSize** [= *long*]
Printer.**dmPaperSizeName(**i%**)**
Printer.**dmPaperSizeX(**i%**)**
Printer.**dmPaperSizeY(**i%**)**

Printer.**dmPaperLength** [= *long*]
Printer.**dmPaperWidth** [= *long*]
Printer.**dmQuality** [= *long*]
Printer.**dmYRes** [= *long*]

Description

To inquire the **Printer** object for device-dependent information, use the **dm**-properties. These properties correspond to the DEVMODE structure, which is internally read by GFA-BASIC 32. By default, these properties return information on the system standard printer.

dmCollate = r/w, %) Determines whether document collation should be done when printing multiple copies: 0 - disabled, 1 - enabled.

dmColor = r/w, %) Determines whether a color printer prints output in color (2) or monochrome (1).

dmCopies = r/w, %) Specifies the number of copies on printers that support multiple-page copies (most laser printers).

dmPaperBin = r/w, $) Indicates the default paper bin on the printer from which paper is fed when printing. To set a paper bin by number use "#1" for upper bin, "#2" for lower bin, "#3" for middle bin, "#4" for manual, "#5" for envelope, "#6" for envelope manual, "#7" for Auto, "#8" for tractor feed, "#9" for small paper feeder, "#10" for large paper bin, "#11" for large capacity feeder, "#14" for attached cassette cartridge.

dmPaperBinName(i) = r, $) Returns the paper bin name for the given number. i% starts at 1, i% = 0 is current printer.

dmPaperSize = r/w, $) Returns or selects the size of the paper to print on. This member can be set to an empty string if the length and width of the paper are both set by the **dmPaperLength** and **dmPaperWidth** properties. Otherwise, the **dmPaperSize** member can be set to a string containing the name of a paper format, like "A4", "A4 (210 x 297 mm)", "Letter 8 1 / 2 x 11", etc. Rather than specifying a name, the size can also set using a number with a leading "#"; "#1" for "Letter", "#9" for DIN-A4, or "#69" for "Japanese Double Postcard 200 x 148 mm" or "#118" for " PRC Envelope #10 Rotated 458 x 324 mm" (PRC = Peoples Republic of China)

dmPaperSizeName(i) = r, $) Returns the name of the size of the paper for the given number. i% starts at 1, i% = 0 is current printer.

dmPaperSizeX(i) = r, %) Returns the horizontal size in tenth of a millimeter for the specified paper format. i% starts at 1, i% = 0 is current printer.

dmPaperSizeY(i) = r, %) Returns the vertical size in tenth of a millimeter for the specified paper format. i% starts at 1, i% = 0 is current printer.

dmPaperLength = r/w, %) Overrides the length of the paper specified by the **dmPaperSize** property, either for custom paper sizes or for devices such as dot-matrix printers, which can print on a page of arbitrary length. These values, along with all other values in this structure that specify a physical length, are in tenths of a millimeter. To initiate, invoke **dmPaperSize** = "" after setting **dmPaperLength**.

dmPaperWidthOverrides the width of the paper specified by the **dmPaperSize** member. To initiate, invoke

dmPaperSize = "" after setting **dmPaperWidth**.

dmQuality = r/w, %) Indicating the printer resolution. -1 = Draft resolution, -2 = Low resolution, -3 = Medium resolution, -4 = High resolution. In addition to the predefined negative values, you can also set *value* to a positive dots per inch (dpi) value, such as 300.

dmYRes = r/w, %) Indicating the printer resolution in y-direction. Some printer drivers allow separate resolutions for horizontal and vertical directions. When a driver supports this, the **dmQuality** sets the x-resolution and **dmYRes** the y-resolution. Values are the same as with **dmQuality**.

Example

```
OpenW 1, 0, 0, 200, 200
Debug.Show
~SetWindowPos(Debug.hWnd, 0, 300, 0, 400, 700,
  $40)
Dim i As Int
Debug "-- PaperBins --" : Print "Paper Bins"
Trace Printer.dmPaperBin
For i = 1 To 14
  Print AT(1, 2); "Processing Bin "; i ; " / 14"
  If Len(Printer.dmPaperBinName(i)) _
    Debug "#";i` Printer.dmPaperBinName(i)
Next
Debug "-- PaperSizes --" : Print AT(1, 4); "Paper
  Sizes"
Trace Printer.dmPaperSize        ' "A4"
Trace Printer.dmPaperSizeX(0)    ' 2100 (x 0.1 mm)
Trace Printer.dmPaperSizeY(0)    ' 2970 (x 0.1 mm)
For i = 1 To 256
  Print AT(1, 5); "Processing Size "; i ; " / 256"
  If Len(Printer.dmPaperSizeName(i)) _
```

```
   Debug "#";i` Printer.dmPaperSizeName(i)
Next
```

Example 2

```
Debug.Show
Trace Printer.hDC
Trace Printer.dmCollate
Trace Printer.dmCopies
Trace Printer.dmColor
Trace Printer.dmPaperSize
Trace Printer.dmPaperLength
Trace Printer.dmPaperWidth
Trace Printer.Orientation
Trace Printer.Zoom
Trace Printer.Duplex
Trace Printer.PaperHeight  // ScaleMode; units
Trace Printer.PaperWidth   // ScaleMode; units
```

Remarks

Setting a printer's **Height** or **Width** property automatically sets **dmPaperSize** to "#256" (user-defined").

The **PaperWidth** and **PaperHeight** properties (read-only) return the size of the paper in **ScaleMode** units.

Other related device mode properties are **Orientation**, **Zoom**, **Duplex**.

To gather printer information before setting the **Printer** object, use **App.PrintInfo**.

See Also

Printer, ShowPageSetup, ShowPrint, SetPrinterByName, PrinterInfo (App), PrinterName (App), Orientation, Zoom,

Duplex

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

Duplex, Orientation, Zoom Properties (Printer)

Purpose

Return or set device mode properties **Duplex**, **Orientation**, and **Zoom**.

Syntax

Printer.**Duplex** [= *long*] *Printer*.**Orientation** [= *long*] *Printer*.**Zoom** [= *long*]

Description

The **Duplex** property returns or sets a value that determines whether a page is printed on both sides (if the printer supports this feature, if not **Duplex** returns 0).

1 (Simplex) - Single-sided printing with the current orientation setting.

2 (Horizontal) - Double-sided printing using a horizontal page turn.

3 (Vertical) - Double-sided printing using a vertical page turn.

The **Orientation** property returns or sets a value indicating whether documents are printed in portrait or landscape mode.

1 (Portrait) - Documents are printed with the top at the narrow side of the paper.

2 (Landscape) - Documents are printed with the top at the wide side of the paper.

The **Zoom** property returns or sets the percentage by which printed output is to be scaled up or down. The default is 0, which specifies that the printed page appears at its normal size. 50 means shrink the output to 50%.

Example

```
Ocx CommDlg cd
cd.Flags = 0
cd.ShowPageSetup
Set Printer = cd
Debug.Show
Trace Printer.Duplex
Trace Printer.Orientation
Trace Printer.Zoom
```

Remarks

The device mode properties **Duplex**, **Orientation**, and **Zoom** are in fact dm-properties, like **dmCollate**, **dmPaperSize**, etc. They could have been named dmDuplex, dmOrientation, and dmZoom, as well. However, for compatibility reasons (VB) they don't.

See Also

Printer, ShowPageSetup, ShowPrint, SetPrinterByName, dm-Properties

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontTransparent, FontUnderline Properties

Purpose

Return or set font styles in the following formats: **Bold**, *Italic*, Strikethru, and <u>Underline</u>.

Syntax

[object**.]FontBold** [= boolean]

[object**.]FontItalic** [= boolean]

[object**.]FontName** [= string]

[object**.]FontSize** [= short]

[object**.]FontStikethru** [= boolean]

[object**.]FontTransparent** [= boolean]

[object**.]FontUnderline** [= boolean]

object:Ocx Object (controls, Form, and Printer)

Description

Use these font properties to format text, either at design time using the Properties window or at run time using code. For **Form** and **Printer** objects, setting these properties doesn't affect graphics or text already drawn on the control or object. For all other controls, font changes take effect on screen immediately.

FontSize takes a Short specifying the size of the font in dots/inch.

FontTransparent returns or sets a Boolean value that determines whether background text and graphics on a **Form** or **Printer** object are displayed in the spaces around characters.

Example

```
Ocx Label lbl = "Example Text", 10, 10, 100, 20
Ocx Command bld = "Bold", 10, 40, 80, 22
Ocx Command itl = "Italic", 100, 40, 80, 22
Ocx Command und = "Underline", 190, 40, 80, 22
Do  : Sleep : Until Me Is Nothing

Sub bld_Click
  If bld.Caption = "Bold"
    bld.Caption = "Not Bold" : lbl.FontBold = True
  Else
    bld.Caption = "Bold" : lbl.FontBold = False
  EndIf
EndSub

Sub itl_Click
  If itl.Caption = "Italic"
    itl.Caption = "Not Italic" : lbl.FontItalic =
      True
  Else
    itl.Caption = "Italic" : lbl.FontItalic = False
  EndIf
EndSub

Sub und_Click
  If und.Caption = "Underline"
    und.Caption = "No Underline" :
      lbl.FontUnderline = True
  Else
    und.Caption = "Underline" : lbl.FontUnderline =
      False
  EndIf
EndSub
```

Remarks

FontTransparent is the same a **GraphMode:**

GraphMode , TRANSPARENT **FontTransparent** = True
GraphMode, OPAQUE **FontTransparent** = False

Note: When changing **FontName**, you may inadvertantly change the **FontSize** as well if the new font does not support the current size: when this happens, GFA-BASIC32 will automatically change **FontSize** to the closest legal for the new font. To prevent any nasty surprises, it is thus advisable to assign the **FontName** first and then specify the **FontSize**.

See Also

Font Object, Font, GraphMode

{Created by Sjouke Hamstra; Last updated: 22/06/2017 by James Gaite}

hDC Property

Purpose

Returns a handle provided by the Microsoft Windows operating environment to the device context of an object.

Syntax

object.**hDC**

object:Form, Printer, CommDlg

Description

This property is a Windows operating environment device context handle. The Windows operating environment manages the system display by assigning a device context for the Printer object and for each form in your application. You can use the **hDC** property to refer to the handle for an object's device context. This provides a value to pass to Windows API calls.

With a **CommDlg** object, this property returns a device context for the printer selected in the Print dialog box when the **cdpReturnDCflag** is set or an information context when the **cdpReturnIC** flag is set.

Example

The following routine copies an area from the Screen.DC and copies it to the hDC of Window 1.

```
// Courtesy of Juergen
```

```
OpenW 1, 0, 0, 402, 402, 0
Win_1.AutoRedraw = 1
BitBlt Screen.GetDC, 400, 400, 400, 400,
  Win_1.hDC, 0,  0, &H00CC0020
```

Remarks

The value of the **hDC** property can change while a program is running, so don't store the value in a variable; instead, use the **hDC** property each time you need it.

See Also

Form, hDC2, hWnd, _DC(), _DC2

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

Page Property (Printer)

Purpose

Returns the current page number when printing.

Syntax

% = *Printer*.**Page**

Description

GFA-BASIC 32 keeps a count of pages that have been printed since your application started or since the last time the **EndDoc** statement was used on the **Printer** object. This count starts at one and increases by one if:

You use the **StartPage** method.

You use **Print** or **Lprint** and the text you want to print doesn't fit on the current page.

Example

```
Dim h As Handle : Global Int32 ct, x, y
Dlg Print Me, 0, h
If h <> 0
  SetPrinterHDC h
  Output = Printer
  FontSize = 12
  StartDoc "test"
  StartPage
  PrintPage()
  EndPage
  StartPage
  PrintPage
  EndPage
  EndDoc
EndIf

Proc PrintPage
  Local p$ = "Page " & Printer.Page
  x = (Printer.Width - TextWidth(p$)) / 2
  y = Printer.Height - TextHeight(p$)
  Text x, y, p$
EndProc
```

See Also

Printer, StartPage, StartDoc, EndDoc

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

PageWidth, PageHeight, PaperWidth, PaperHeight Properties

Purpose

Return the page and paper size in **ScaleMode** units.

Syntax

Printer.**PageWidth**

Printer.**PageHeight**

Printer.**PaperWidth**

Printer.**PaperHeight**

Description

The return type of these properties is Single.

PageWidth and **PageHeight** return the size of printable area in **ScaleMode** units.

PaperWidth and **PaperHeight** return the size of the paper in **ScaleMode** units.

Example

```
Debug.Show
SetPrinterHDC Printer.hDC
Trace Printer.ScaleMode
Trace Printer.PageHeight
Trace Printer.PageWidth
Trace Printer.PaperHeight
Trace Printer.PaperWidth
Trace Printer.dmPaperSize     ' A4
Trace Printer.dmPaperSizeX(0) ' 2100 (x 0.1 mm)
Trace Printer.dmPaperSizeY(0) ' 2970 (x 0,1 mm)
```

Remarks

The device mode properties **dmPaperSizeX** and **dmPaperSizeY** return the size in tenths of a millimeter.

See Also

Printer

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

PrintScroll Property

Purpose

Returns or sets the scrolling behaviour of a **Form** or **Printer**.

Syntax

[object.]**PrintScroll** [= value]

object:Form or Printer
value:Boolean exp

Description

Determines the scrolling behaviour of the output device when the current output position has reached the bottom of the output area. Used without an object will change the current output object.

Example

```
OpenW 1
PrintScroll = 1
AutoRedraw = 1
Local i As Int
For i = 0 To 100
  Print i
  Pause 0.5
Next i
```

See Also

Form, Printer

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

PrintWrap Property

Purpose

Returns or sets line wrapping with Print of a **Form** or **Printer**.

Syntax

[object.]**PrintWrap** [= value]

object:Form or Printer
value:Boolean exp

Description

Determines the wrapping of a string on the output device when the string is about to cross the right boundary. The string is checked to fit in the rest of the line. When the string doesn't fit it will be completely printed on the next line. First a CRLF is invoked, and maybe a NewPage on the Printer, and then the string is printed.

Example

```
OpenW Full 1
PrintWrap = 1
Local n As Int32
For n = 1 To 3000
  Print n; " ";
Next n
```

Remarks

With PrintWrap = True and a string *a* to be printed, the check is like:

```
If CurrentX + TextWidth(a) > ScaleWidth Then Print
Print a
```

See Also

Form, Printer

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

AbortDoc Command

Purpose

Cancels a printer job.

Syntax

[*Printer*.]**AbortDoc**

Description

AbortDoc stops the current print job and erases everything drawn since the last call to the **StartDoc** function.

Applications should call the **AbortDoc** function to stop a print job if an error occurs, or to stop a print job after the user cancels that job. To end a successful print job, an application should call the **EndDoc** function.

Example

```
OpenW 1
Local x%, o
Output = Printer
StartDoc "GFA32 Test"
StartPage
Print "GFA"
Circle 200, 200, 150
Box 350, 200, 450, 300
EndPage
StartPage
Print "GFA2"
EndPage
```

```
AbortDoc
EndDoc
Output = Win_1 : Print "finished"
KeyGet x% : CloseW 1 : End
```

Remarks

A printer job is most often finished before it is even started and is mostly canceled through the printer tray icon. Maybe **AbortDoc** has some use, though.

Another way of aborting a printer job from a program is by using the Printer object event sub **Printer_AbortProc**.

See Also

Printer StartDoc, EndDoc, NewFrame, StartPage

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

StartDoc, EndDoc Commands

Purpose

Starts a new printer job on a printer device.

Syntax

[*Printer*.]**StartDoc** name [, o]

[*Printer*.]**EndDoc**

name:sexp
o:Variant variable

Description

StartDoc is a method of the **Printer** object. **StartDoc** must be invoked to start a printer job and is used before **StartPage**. *name* can be any text string used to identify the job. This name will be shown in the Printer Manager of the operation system during the output. A printer job is closed by invoking **EndDoc**.

When used without the Printer object the output must be redirected first to the printer using **Output =** .

An output device can be a printer, a fax, a copy machine, etc. any device, the operating system allow to use as a printer.

The optional Variant variable o is undocumented and is not mirrored in the GDI Printer API function of the same name.

Example

```
OpenW 1
Local h As Handle
Dlg Print Win_1, 40, h
If h <> 0
  SetPrinterHDC h
  // output to the standard device
  Output = Printer : FontSize = 12
  StartDoc "test" : StartPage
  Print "GFA"
  Circle 200, 200, 150
  Box 350, 200, 450, 300
  EndPage : StartPage
  Print "GFA2"
  EndPage : EndDoc
  Output = Win_1 : Print "Printing finished"
Else
  Print "Printing Cancelled"
EndIf
```

Remarks

StartDoc starts a print job. **StartPage** prepares the printer driver to accept data. **EndPage** informs the device that the application has finished writing to a page. The printer spooler realizes the output on the printer. Repeat **StartPage/EndPage** for the next page. **EndDoc** ends a print job.

When the printer is used a line printer with **Lprint,** the printer job is automatically created. Therefore, **Lprint "";** is the same as **StartDoc ""/StartPage.**

See Also

Printer, StartDoc, EndPage, StartPage

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

StartPage, EndPage Commands

Purpose

Starts a new page on a printer device.

Syntax

[*Printer*.]**StartPage**

[*Printer*.]**EndPage**

Description

StartPage is a method of the **Printer** object. **StartPage** must be invoked before printing and is used after a previous **EndPage**. An output device can be a printer, a fax, a copy machine, etc. any device, the operating system allow to use as a printer.

EndPage is used to mark the end of the page.

Example

```
OpenW 1
Local h As Handle
Dlg Print Win_1, 40, h
If h <> 0
  SetPrinterHDC h
  // output to the standard device
  Output = Printer : FontSize = 12
  StartDoc "test" : StartPage
  Print "GFA"
  Circle 200, 200, 150
  Box 350, 200, 450, 300
  EndPage : StartPage
  Print "GFA2"
  EndPage : EndDoc
  Output = Win_1 : Print "Printing finished"
Else
  Print "Printing Cancelled"
EndIf
```

Remarks

StartDoc starts a print job. **StartPage** prepares the printer driver to accept data. **EndPage** informs the device that the application has finished writing to a page. The printer spooler realizes the output on the printer. Repeat **StartPage**/**EndPage** for the next page. **EndDoc** ends a print job.

See Also

Printer, Page, EndDoc, StartDoc, StartPage

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

NewFrame Command

Purpose

Ends and starts a new page on a printer device.

Syntax

[*Printer*.]**NewFrame**

Description

NewFrame is a method of the **Printer** object. **NewFrame** is an old form for **EndPage** & **StartPage**. It is advised to use **EndPage/StartPage**, rather than **NewFrame**.

Example

```
OpenW 1
Local h As Handle
Dlg Print Win_1, 0, h
SetPrinterHDC h
Output = Printer : FontSize = 12
StartDoc "test" : StartPage
Print "GFA"
Circle 200, 200, 150
Box 350, 200, 450, 300
NewFrame      ' EndPage : StartPage
Print "GFA2"
EndPage : EndDoc
Output = Win_1
```

Remarks

StartDoc starts a print job. **StartPage** prepares the printer driver to accept data. **EndPage** informs the device that the application has finished writing to a page. The printer spooler realizes the output on the printer. Repeat **StartPage**/**EndPage** for the next page. **EndDoc** ends a print job. The **Page** property returns the current page number.

See Also

Printer, EndDoc, StartDoc, EndPage, Page, StartPage

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

TextHeight, TextWidth Method

Purpose

Determines a string's height and width in **ScaleMode** units.

Syntax

x! = [Object.]**TextHeight**(a$)

x! = [Object.]**TextWidth**(a$)

Object:Ocx Object
x!: Single
a$: sexp

Description

Returns the height and width of a character expression with respect to the average character width for the font of Ocx object *Object*. Without a specified *Object* the current active output object is used (**Form** or **Printer**).

Example

```
Local a$ = "Hello World", tx%(2, 2)
OpenW 1, , , 400, 200
AutoRedraw = 1
FontName = "Arial" : FontSize = 20
tx(1, 1) = TextHeight(a$), tx(1, 2) =
  TextWidth(a$)
ScaleMode = basTwips
```

```
tx(2, 1) = TextHeight(a$), tx(2, 2) =
  TextWidth(a$)
FontSize = 9
Print "Height in Pixels: "; tx(1, 1)
Print "Width in Pixels: "; tx(1, 2)
Print "Height in Twips: "; tx(2, 1)
Print "Width in Twips: "; tx(2, 2)
FontSize = 20
ScaleMode = basPixels
Line 0, 100, 300, 100
Text 50, 100 - TextHeight(a$) / 2, a$
```

Remarks

See Also

Form, ScaleMode

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

AbortProc Event (Printer)

Purpose

Sets the application-defined abort function that allows a print job to be canceled during spooling.

Syntax

Sub **Printer_AbortProc**(*hDC%, error%, Cancel?*)

Description

Before you start a print job, you can establish an abort procedure simply by including **Printer_AbortProc** sub event into your GFA-BASIC 32 application. All the necessary initialization is performed automatically by GFA-BASIC 32..

GDI calls the **AbortProc** every 2 seconds during a print job to inform the application of spooler errors and to allow the application to abort the job when desired. GDI calls the **AbortProc** function with information about why it is being called; this value is either an error code from the spooler or zero, which indicates that the function is being called simply to allow an abort.

hDC% Handle to the device context for the print job.

error% Specifies whether an error has occurred. This parameter is zero if no error has occurred; it is SP_OUTOFDISK if Print Manager is currently out of disk space and more disk space will become available if the application waits.

*Cancel?*Return TRUE to cancel the print job, or False to continue.

Example

```
Lprint
Printer.EndPage  ' Error condition
Printer.EndDoc
Do
  Sleep
Loop

Sub Printer_AbortProc(hDC%, iError%, Cancel?)
  Debug hDC%; iError%
  Cancel = 1
EndSub
```

Remarks

Most applications give the user an opportunity to abort a print job by providing a dialog box with a cancel button or a variation on that theme. An application can use several time slices during a print job to check for a user cancellation of the printing. When GDI calls the **AbortProc** function, the printing process is yielding to the application for exactly such purposes; this is a good opportunity to check for user input. When the application itself is performing a time-intensive operation, it can yield to the abort-checking code when desired.

When using a Cancel dialog box add DoEvents to the Printer_AbortProc so that the application can process the mouse message.

See Also

Printer, AbortDoc

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

AutoNewFrame Event (Printer)

Purpose

Occurs when the printer needs a form feed as a result of **Lprint**, or **Print** (also **Dir**, **Files**) when **Output** = **Printer**.

Syntax

Function Printer_AutoNewFrame() As Int

Description

This event is a **Function**, rather than a **Sub**. The function return value (<> 0) can be used to suppress the auto form feed. The function AutoNewFrame can be used to print multiple columns by setting **Printer**.**Left** and **Printer**.**Width**.

Within this function the **Output** = **Printer**.

Example

Print **Dir** in multiple columns:

```
Lprint "";
Global Int PrintColumns
Global Int PrintColIndex
Global Float PrintColWidth
PrintColWidth = Printer.TextWidth("XXXXXXXXX.XXX")
PrintColumns = Printer.DefWidth Div PrintColWidth
PrintColIndex = 0
```

```
Debug.Print PrintColumns
Printer.Width = PrintColWidth - 4
Local i%
Output = Printer
Dir
Output = Me

Function Printer_AutoNewFrame() As Int
  PrintColIndex++
  If PrintColIndex >= PrintColumns
    PrintColIndex = 0
    Printer.Left = Printer.DefLeft
    Printer.Width = PrintColWidth - 4
    Printer.CurrentY = 0
    Printer.CurrentX = 0
    Return 0
  End If
  Printer.Left = PrintColWidth * (PrintColIndex -
    1) + Printer.DefLeft
  Printer.Width = PrintColWidth - 4
  Printer.CurrentY = 0
  Printer.CurrentX = 0
  Return 1
EndFunc
```

Print **Dir** in multiple columns with a page header.

```
Lprint "";
Global Int PrintColumns
Global Int PrintColIndex
Global Float PrintColWidth
PrintColumns = 5
PrintColWidth = Printer.DefWidth / PrintColumns
PrintColIndex = 0
Printer.Width = PrintColWidth - 4
Printer.Height = 200
Local i%
```

```
Output = Printer
For i = 0 To 90 Step 10
  Line 0, 0, Printer.Width, i
Next i
CurrentY = 0 : CurrentX = 0
Dir
Output = Me

Function Printer_AutoNewFrame() As Int
  Local i%
  PrintColIndex++
  If PrintColIndex >= PrintColumns
    PrintColIndex = 0
    NewFrame
    Printer.Left = Printer.DefLeft
    Printer.Width = PrintColWidth - 4
    For i = 0 To 90 Step 10
      Line 0, 0, Printer.Width, i
    Next i
    Printer.CurrentY = 0
    Printer.CurrentX = 0
    Return 1
  End If
  Printer.Left = PrintColWidth * PrintColIndex +
    Printer.DefLeft
  Printer.Width = PrintColWidth - 4
  For i = 0 To 90 Step 10
    Line 0, 0, Printer.Width, i
  Next i
  Printer.CurrentY = 0
  Printer.CurrentX = 0
  Return 1
EndFunc
```

The automatic Form feed is disabled (always Return 1) and replaced by a hard coded **NewFrame** when the number of columns is exceeds 5.

Remarks

Note The print job should be ended with **EndPage**/**EndDoc** otherwise the spooler file isn't closed before the end of the program.

The **Page** property returns the current page number.

See Also

Printer, StartDoc, StartPage, EndPage, Page, EndDoc, NewFrame

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

SetPrinterHDC and SetPrinterByName Commands

Purpose

Sets a printer by its HDC or name.

Syntax

SetPrinterByName p$

SetPrinterHDC h

p$	*: sexp*
h	*: handle*

Description

SetPrinterHDC allows the printer to be set using a printer object's DC handle. This is particularly useful with the Dlg Print command.

SetPrinterByName allows for selecting a printer without using a common dialog box. The printer name *p$* must exist, if not an error is generated. To be sure, use this command in a **Try**/**Catch** structure.

A list of existing printers can be obtained using **App**.**PrinterCount** and **App**.**PrinterName**().

Example

```
Local h As Handle, n As Int32
OpenW 1
FontBold = True : Text 10, 10, "Select a Printer:"
  : FontBold = False
Ocx ListBox lb = "", 10, 25, 250, 100
For n = 1 To App.PrinterCount
  lb.AddItem App.PrinterName(n), n
Next n
Ocx Command cmd = "Select", 85, 140, 100, 22 :
  cmd.Enabled = False
Do : Sleep : Until Win_1 Is Nothing
OpenW Hidden 1
Dlg Print Win_1, 0, h
If h <> 0
  SetPrinterHDC h
  // For some reason, DeviceName is returned as
    blank
  Message "Printer Selected:" & #13#10 &
    Printer.DeviceName
  PrintOut
EndIf
CloseW 1

Sub cmd_Click
  SetPrinterByName lb.List(lb.ListIndex)
  Message lb.List(lb.ListIndex) & " selected as
    current printer"
  PrintOut
  Win_1.Close
EndSub

Sub lb_Click
  If lb.ListIndex <> -1 Then cmd.Enabled = True
EndSub

Sub PrintOut
  Output = Printer
  StartDoc "Test"
  StartPage
  FontSize = 12
  Print "Success!"
  EndPage
  EndDoc
  Output = Win_1
EndSub
```

Remarks

The SetPrinter... commands only temporarily select a printer and do not change which printer is considered default. If the aim is to change to permamently change the default printer, a variation on the following code can be used:

```
Local obj As Object
Set obj = CreateObject("WScript.Network")
~obj.SetDefaultPrinter("EPSON BX305 Plus Series")
Set obj = Nothing
```

Known Issues

Prior to OCX v2.33/2.34, using **SetPrinterByName** could cause a buffer overflow as GFA BASIC 32 did not reserve enough memory for the information received from some more modern printers; there is no workaround for this (barring using **Try**/**Catch** to catch the error) so the only fix is to download the latest version of GfaWin23.ocx.

See Also

Printer, CommDlg, App

{Created by Sjouke Hamstra; Last updated: 04/03/2018 by James Gaite}

PrinterCount, PrinterName, PrinterInfo Properties (App)

Purpose

These properties enable you to gather information about all the available printers on the system.

Syntax

%= *App*.**PrinterCount**

$ = *App*.**PrinterName**(i%)

v = *App*.**PrinterInfo**(*Printer$, What$*)

v:Variant

Description

The **PrinterCount** property returns the number of printers available.

The **PrinterName(i)** property returns the name of the printer with index *i%*.

The **PrinterInfo(***Printer$, What$***)** property returns the device mode information *What$* of the printer with name *Printer$*. Many printer features are device dependent, For example, not all printers support all paper sizes or support landscape printing. You have to actually check the parameters. This device-dependent information is found in the DEVMODE structure. The *What$* parameter takes (some of) the DEVMODE members as a string and **PrinterInfo** returns the value as a Variant. The following entries can be used for *What$*:

```
Dim p$ = App.PrinterName(1)
Trace App.PrinterInfo( p$, "Driver")
Trace App.PrinterInfo( p$, "Orientation")
Trace App.PrinterInfo( p$, "PaperSize")
Trace App.PrinterInfo( p$, "PaperLength")
Trace App.PrinterInfo( p$, "PaperWidth")
Trace App.PrinterInfo( p$, "Copies")
Trace App.PrinterInfo( p$, "Quality")
Trace App.PrinterInfo( p$, "Color")
Trace App.PrinterInfo( p$, "Duplex")
Trace App.PrinterInfo( p$, "PaperBin")
```

Example

The following code searches all available printers to locate the first printer with its page orientation set to portrait, then sets it as the default printer:

```
Dim pr$, i%
For i = 0 To App.PrinterCount - 1
  pr$ = App.PrinterName(i)
  Print pr$
  If App.PrinterInfo(pr$, "Orientation") = 1
    SetPrinterByName pr$
    Exit For
  EndIf
Next
```

Remarks

SetPrinterByName implicitly invokes **Set Printer =**.

GFA-BASIC 32 does not provide a ***Printers*** collection, instead you should use the **App** properties to enumerate the printers attached to system.

See Also

App, Printer, SetPrinterByName

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Align, Orientation Property

Purpose

Returns or sets a value that determines whether an object is displayed in any size anywhere on a form or whether it's displayed at the top, bottom, left, or right of the form and is automatically sized to fit the form's width.

Syntax

object.**Align** [= *number*]

object2.**Orientation** [= *number*]

object:Toolbar, StatusBar, Scroll, Slider, ProgressBar, Form Ocx Object
object2:Scroll, Slider, ProgressBar
number:iexp

Description

Use the **Align** property to position an Ocx control or Ocx Form at the border of its parent. The **Orientation** property sets a value (**basHorizO** or **basVertO**) that determines whether the control is oriented horizontally or vertically. The **Align** property always overrules the **Orientation** property, as well as reseting any changes previously made to **Width** or **Height**.

An **Ocx Form** can be used in a MDI parent (ParentW), which only allows Ocx controls that can align themselves to a border. These are **Toolbar**, **StatusBar**, **Scroll**, **Slider**, **ProgressBar**, and **Form Ocx**.

The following AlignBarConst values are allowed:

Value	AlignBarConst	Meaning
0	**basNoAlign**	None - size and location can be set at design time or in code.
1	**basTop**	Top - object is at the top of the form, and its width is equal to the form's **ScaleWidth** property setting.
2	**basBottom**	Bottom - object is at the bottom of the form, and its width is equal to the form's **ScaleWidth** property setting.
3	**basLeft**	Left - object is at the left of the form, and its width is equal to the form's **ScaleWidth** property setting.
4	**basRight**	Right - object is at the right of the form, and its width is equal to the form's **ScaleWidth** property setting.

Ocx controls that are aligned at the border of the parent form (using **Align**) change the **Scale** settings of the parent. **ScaleLeft = 0** and **ScaleTop = 0** are set to the top-left pixel of the uncovered client area of the form using *SetViewportOrgEx* API. **ScaleWidth** and **ScaleHeight** are set to width and height of the uncovered area. The mouse coordinates returned from **MouseX**, **MouseY** and that are passed in the forms **MouseMove**, **MouseUp**, and **MouseDown** events are relative to the new origin.

Example

```
Debug.Show
```

```
ParentW 1
Ocx ToolBar tb
Ocx StatusBar st
Ocx Form ofrm = , , , 200, 10
ofrm.Align = basLeft
Set Me = Win_1
Trace Me.ScaleTop
Trace Me.ScaleLeft
Trace Me.ScaleHeight : Trace _Y
Trace Me.ScaleWidth : Trace _X
Do
  Sleep
Until Me Is Nothing
```

Remarks

The **Align** property overrules the **Orientation** property (Slider).

See Also

Form, ToolBar, StatusBar, Scroll, Slider, ProgressBar

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Appearance Property

Purpose

Returns or sets the paint style of Ocx controls or Form.

Syntax

Object.**Appearance** [= value]

Object:Ocx Object
value:iexp (0..15)

Description

The **Appearance** property influences the border of the form and Ocx controls. The value ranges from 0 to 15, which represents a combination of 4 bits. Effectively, the following WS_EX window styles are applied.

Value	Bit	Style Effect
0	--	Flat
11	WS_EX_CLIENTEDGE	Specifies that a window has a 3D look - that is, a border with a sunken edge.
22	WS_EX_WINDOWEDGE	Specifies that a window has a border with a raised edge.
43	WS_EX_STATICEDGE	Creates a window with a three-dimensional border style intended to be

		used for items that do not accept user input.
84	WS_EX_DLGMODALFRAME	Designates a window with a double border that may (optionally) be created with a title bar when you specify the WS_CAPTION style.

The appearance of the OCX can be adjusted further by setting the **BorderStyle** property. **BorderStyle** = 2 draws a thick border.

For a **Label** Ocx the **Appearance** property simply applies optical effects. Setting the **BorderStyle** to 2 doubles the border line.

For a **Checkbox** Ocx only **basFlat** (0) and **basThreeD** (1) are allowed values for **Appearance**.

Flat Scroll Bar

By setting the **Appearance** property, the Scroll OCX is changed to a flat scroll bar (equivalent to the VB ***FlatScrollBar*** control). The flat scrollbar control is a mouse-sensitive version of the standard Windows scroll bar that offers two-dimensional formatting options. It can also replace the standard Windows three-dimensional scroll bar. With the ***FlatScrollBar*** you can disable either of the scroll arrows, this provides additional feedback to the user as an indication to scroll in a particular direction based on other factors in the program.

Appearance can have following values:

0A normal, non flat scroll bar is displayed. No special visual effects will be applied (FSB_REGULAR_MODE).

1A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar will be displayed in 3-D (FSB_ENCARTA_MODE).

2A standard flat scroll bar is displayed. When the mouse moves over a direction button or the thumb, that portion of the scroll bar will be displayed in inverted colors (FSB_FLAT_MODE).

Example

```
Form test
Me.Appearance = 15
Do : Sleep : Until Me Is Nothing
```

See Also

Form, BorderStyle

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Cancel, Default Property (Command)

Purpose

Returns or sets a value indicating whether a command button is the Cancel or Default button on a form.

Syntax

Command.**Cancel** [= Boolean]

Command.**Default** [= Boolean]

Description

Only one **Command** button on any one form can be the Cancel button so, when the **Cancel** property is set to **True** for one **Command**, it's automatically set to **False** for all other **Command** controls on the same form. Similarly with the **Default** property: only one **Command** can have the **Default** property set. However, it is possible that one button can be both the **Cancel** AND **Default** control.

A **Command** button with either of these properties set can be selected just like any other button: by clicking it or pressing ENTER when the control has focus. However, in addition, these **Command** controls can be activated even when they do not have the focus by pressing ESC for the **Cancel** button and ENTER (if no other **Command** has the focus) for the **Default**.

Example

```
OpenW 1
Ocx Command cmdOk = "OK", 20, 20, 50 * 2, 14 * 2
.Default = True
Ocx Command cmdCancel = "Cancel", 20, 50, 50 * 2,
  14 * 2
.Cancel = True
Do
  Sleep
Until Me Is Nothing

Sub cmdOk_Click
  MsgBox "OK button selected"
EndSub

Sub cmdCancel_Click
  MsgBox "Cancel button selected"
  Win_1.Close
EndSub
```

Remarks

When you have a dialog box with an OK and/or Cancel button, do not give the keys accelerators. The dialog manager already has those buttons covered. The hotkey for the OK button is Enter (since it is the default pushbutton), and the hotkey for the Cancel button is ESC (since its ID is IDCANCEL).

Of course that during the lifetime of a dialog box, the default pushbutton may change, but the principle still stands: Do not give the OK button a keyboard accelerator.

Finally, don't forget that the recommended minimum size for pushbuttons is 50 dialog units by 14 dialog units.

See Also

Command

{Created by Sjouke Hamstra; Last updated: 11/10/2017 by James Gaite}

Caption Property

Purpose

Determines the text displayed in the Ocx object.

Syntax

object.**Caption** [= *string*]

Description

For a Form, determines the text displayed in the Form's title bar. When the form is minimized, this text is displayed below the form's icon.

For a control **Caption** determines the text displayed in or next to a control.

You can use the **Caption** property to assign an access key to a control. In the caption, include an ampersand (&) immediately preceding the character you want to designate as an access key. The character is underlined. Press the ALT key plus the underlined character to move the focus to that control. To include an ampersand in a caption without creating an access key, include two ampersands (&&). A single ampersand is displayed in the caption and no characters are underlined.

Example

```
Ocx Command cmd = "OK", 10, 10, 80, 22
Do : Sleep : Until Me Is Nothing
```

```
Sub cmd_Click
  cmd.Caption = (cmd.Caption = "OK" ? "Not OK" :
    "OK")
EndSub
```

Remarks

When you create a new object, its default caption is the default **Name** property setting. This default caption includes the object name and an integer, such as Command1 or Form1. For a more descriptive label, set the **Caption** property.

See Also

Form, Text

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Font Property

Purpose

Returns or sets a **Font** object.

Syntax

object.**Font** [= font]

object: visible OCX objects

Description

Use the **Font** property of an object to identify a specific **Font** object whose properties you want to use. The **Font** property can also be used to assign a new Font object to the Ocx object. Usually, assigning a new object to an object variable requires the **Set** command.

For example, the following code changes the **Bold** property setting of a **Font** object identified by the **Font** property of a **TextBox** object.

Example

```
OpenW 1 ': Win_1.Font.Name = "Courier New"
Text 10, 10, "This is the native Window font"
Ocx Label lbl = "This is the Label font", 10, 25,
  200, 15 : lbl.Font.Name = "Times New Roman"
Ocx Command cmd1 = "Match Font to that of Window",
  15, 45, 90, 36 : cmd1.Font.Name = "Arial" :
  cmd1.WinStyle = cmd1.WinStyle | BS_MULTILINE
```

```
Ocx Command cmd2 = "Make Label Bold", 115, 45, 90,
  36 : cmd2.Font.Name = "Arial" :  cmd2.WinStyle =
  cmd2.WinStyle | BS_MULTILINE
Do : Sleep  : Until Win_1 Is Nothing

Sub cmd1_Click
  If lbl.FontName = "Times New Roman" //
    lbl.Font.Name and lbl.FontName are
    interchangeable
    Set lbl.Font = Win_1.Font  // this copies all
      Font characteristics, not just Name and resets
      Bold if set, so...
    cmd2.Caption = "Make Label Bold"
    cmd1.Caption = "Restore original font to Label"
  Else
    lbl.FontName = "Times New Roman"
    cmd1.Caption = "Match Font to that of Window"
  EndIf
EndSub

Sub cmd2_Click
  If lbl.Font.Bold = True
    lbl.FontBold = False // lbl.Font.Bold and lbl.
      FontBold are interchangeable
    cmd2.Caption = "Make Label Bold"
  Else
    lbl.Font.Bold = True
    cmd2.Caption = "Make Label Normal Weight"
  EndIf
EndSub
```

Remarks

The preferred way to setting font attributes is by using a particular font attribute property, like **FontBold**. The generated code is smaller and faster for each saved dot.

See Also

Font Object, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontTransparent, FontUnderline, SetFont.

{Created by Sjouke Hamstra; Last updated: 06/10/2014 by James Gaite}

Height, Width Properties

Purpose

Return or set the dimensions of an Ocx object.

Syntax

object.**Height** [= value]

object.**Width** [= value]

object:Ocx objects
value:Single exp

Description

The **Height** and **Width** properties set the outer dimensions of an OCX control or Form. The *value* is specified in pixels. For OCX controls, the units can be adjusted to the current scaling of the parent Form. The Form property **OcxScale** = True sets the coordinate scheme for the Ocx controls to the **ScaleMode** of the Form. By default the **ScaleMode** = **basPixels** (in VB mostly twips).

For the **Screen** object they return the height and width of the screen.

For the **Printer** object the physical dimensions of the paper set up for the printing device. If set, values in these properties are used instead of the setting of the **PaperSize** property

Example

```
Form Frm
Do
  Sleep
Until Me Is Nothing

Sub Frm_Click ()
  With Frm
    .Width = Screen.Width * .75           ' Set width
     of form.
    .Height = Screen.Height * .75         ' Set
     height of form.
    .Left = (Screen.Width - .Width) / 2   ' Center
     form horizontally.
    .Top = (Screen.Height - .Height) / 2 ' Center
     form vertically.
  End With
End Sub
```

This example sets the size of a form to 75 percent of screen size and centers the form when it is loaded.

Remarks

See Also

Form, Left, Top, Move, OcxScale, ScaleMode

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

HelpContextID, WhatsThisHelpID Properties

Purpose

Both properties return or set an associated context number for an object. Use **HelpContextID** to provide context-sensitive Help for your application and use **WhatsThisHelpID** to provide context-sensitive Help for your application using the What's This pop-up in Windows Help.

Syntax

object.**HelpContextID** [= *number*]

object.**WhatsThisHelpID** [= *number*]

object:Any Ocx object, except ImageList, Timer and CommDlg
number:Long exp

Description

The **HelpContextID** property is used to link a user interface element (such as a control, form, or menu) to a related topic in a Help file. The **HelpContextID** property must be a Long that matches the Context ID of a topic in a WinHelp (.hlp) or HTML Help (.chm) file.

The **WhatsThisHelpID** property stores a value that is used to provide context-sensitive Help for your application using the What's This pop-up. What's This Help provides quick access to Help text in a popup window without the need to open the Help viewer. It is typically used to provide simple assistance for user interface elements such as controls. The property can be used in the **OnCtrlHelp** event sub which is invoked when the What's This mouse cursor [?] is clicked on an Ocx object.

If an object does not have either the **HelpContextID** or the **WhatsThisHelpID** property - all non-Ocx Control objects for example - then you can try to attach one using *retval* = **SetWindowContextHelpId**(*hWnd*, *HelpContextID*), where *hWnd* is the handle for the control or object; *retval* is True if the API succeeded.

Setting the HelpButton property of a form to True enables What's This Help.

For displaying help derived from an older WinHelp32.exe (.hlp) file, you can use the ShowHelp method of the CommDlg object to execute Windows Help; to access a newer HTMLHelp file, see the example in Accessing HTML Help Files.

Example

See the example for OnHelp.

Remarks

When these properties aren't used for a help file IDs, they can be used as custom 32-bit integer values, like **Tag**. **Tag** is a string property and as such is more time consuming than a 32-bit integer.

See Also

Form, OnCtrlHelp, Tag, ShowHelp

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Index, Key Property

Purpose

Returns or sets the number or string that uniquely identifies an object in a collection.

Index returns or sets the number that uniquely identifies a control or form in an array. Available only if the control or form is part of a control or form array.

Syntax

Object.**Index** [= value]

Object.**Key** [= value]

Object:Form, Animation, Button, CheckBox, ColumnHeader, ComboBox, Command, Frame, Image, Label, ListBox, ListImage, ListItem, ListView, MenuItem, MonthView, Node, Option, Panel, Progress, RichEdit, Scroll, Slider, Tab, TabStrip, Textbox, Timer, TrayIcon, TreeView, UpDown Object

Description

Ocx Array - For an Ocx control or form the **Index** property returns a number that uniquely identifies the object in a control or form array.

The control or form array is a group of controls or forms that share common names, types, and event procedures. Each control or form has a unique index. When a control or form in the array recognizes an event, it calls the event procedure for the group and passes the index as an argument, allowing your code to determine which control or form recognized the event. For example:

```
OpenW 44
Ocx Command cmd(1) = "But1", 10, 10, 50, 20
Ocx Command cmd(289) = "But289", 10, 40, 50, 20
Print Form(44).Index // = 0 Doesn't return 44 as
  expected
Print cmd(289).Index // = 289
Do : Sleep : Until Form(44) Is Nothing

Sub cmd_Click(Index%)
  Message "Index No:" & Index% & #13#10 & "Form
    Index:" & Me.Index
EndSub
```

Collections - For collections the Index property returns the value of the order of the object in the collection. The **Index** property is set by default to the order of the creation of objects in a collection. The index for the first object in a collection will always be one (1). The order of an object can change when objects in the collection are reordered, such as when you set the **Sorted** property to **True**. If you expect the **Index** property to change dynamically, it may be more useful to refer to objects in a collection by using the **Key** property.

The **Key** property returns or sets a string that uniquely identifies a member in a collection. You can set the **Key** property when you use the **Add** method to add an object to a collection, but you can change the name afterwards.

Example

```
Global Enum sbrText = 0, sbrFlat, sbrRaise,
  sbrCaps, sbrNum, sbrScroll, sbrIns, sbrDate
```

```
Dim p As Panel
Ocx StatusBar sb
sb.Panels.Add , "Part1", "Part 1", sbrText
sb.Panels.Add , "Caps", "Caps", sbrCaps
sb.Panels.Add , "Num", "Num", sbrNum
sb.Panels.Add , "Scroll", "Scroll", sbrScroll
sb.Panels.Add , "INS", "INS", sbrIns
sb.Panels.Add , "Date", "c", sbrDate
sb.Item("Date").Text = "AM/PM"
For Each p In sb
  Print p.Key, p.Index
Next
```

See Also

Form, Animation, Button, CheckBox, ColumnHeader, ComboBox, Command, Frame, Image, Label, ListBox, ListImage, ListItem, ListView, MenuItem, MonthView, Node, Option, Panel, Progress, RichEdit, Scroll, Slider, Tab, TabStrip, Textbox, Timer, TrayIcon, TreeView, UpDown

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

WinStyle, PushLike, ThreeState, Value Properties

Purpose

WinStyle returns or sets the window style of a **Command**, **Option**, and **CheckBox**.

PushLike returns or sets the pushed style state of a **Command**, **Option,** or **CheckBox**.

ThreeState returns or sets the BS_3STATE of a **CheckBox**.

Value returns or sets the state of the control.

Syntax

object.**WinStyle** [= value]

object.**PushLike** [= boolean]

object.**ThreeState** [= boolean]

object.**Value** [= long]

object:Ocx object

Description

Returns or sets the actual windows style of the control. For **Command**, **Option**, and **CheckBox** controls the WS_* window style API constants and the BS_* button styles can be used.

The **PushLike** property sets the BS_PUSHLIKE style and makes a button (such as a check box, three-state check box, or radio button) look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked.

The **ThreeState** property sets the BS_3STATE button style and creates a check box that can be grayed as well as checked or unchecked. Use the grayed state to show that the state of the check box is not determined.

The **Value** property can have the following values:

- **CheckBox** control - 0 is Unchecked (default), 1 is Checked, and 2 is Grayed (dimmed).

- Command control - 1 indicates the button is chosen; 0 (default) indicates the button isn't chosen. Setting the **Value** property to 1 in code invokes the button's Click event.

- Option control - 1 indicates the button is selected; 0 (default) indicates the button isn't selected

Example

```
Ocx Command cmd = "This is a"#10"Multiline
  Button", 10, 10, 140, 40 : cmd.WinStyle =
  cmd.WinStyle | BS_MULTILINE
Ocx CheckBox chk = "This is a 3-State CheckBox",
  10, 60, 200, 14 : chk.ThreeState = True :
  chk.Value = 2
Ocx Option opt1 = "This is a normal Option
  control", 10, 90, 200, 14 : .Value = 1
Ocx Option opt2 = "This is a PushLike Option
  control", 10, 105, 200, 22 : opt2.PushLike = True
Do : Sleep : Until Me Is Nothing
```

Remarks

Be careful, changing window styles is not always without errors.

See Also

Command, CheckBox, Option

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

TabStop Property

Purpose

Returns or sets a value indicating whether a user can use the TAB key to give the focus to an object.

Syntax

*object***.TabStop** [**=** *boolean*]

object:Ocx object

Description

Designates the object as a tab stop (default). When set to False the object is bypassed when the user is tabbing, although the object still holds its place in the actual tab order, as determined by Ocx View window.

Example

```
Ocx Command cmd1 = "Tab Stop", 10, 10, 120, 22
Ocx Command cmd2 = "No Tab Stop", 140, 10, 120, 22
  : .TabStop = False
Ocx Command cmd3 = "Tab Stop Again", 270, 10, 120,
  22
Ocx Command cmd4 = "Close Window"#10"No Tab Stop",
  140, 50, 120, 40 : .TabStop = False : .WinStyle =
  .WinStyle | BS_MULTILINE
Do : Sleep : Until Me Is Nothing

Sub cmd4_Click
  Me.Close
```

```
EndSub
```

See Also

Form

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

Text, TextLength Property

Purpose

Text returns or sets the text contained in an object. **TextLength** returns the number of characters in a **RichEdit** control.

Syntax

Object.**Text** [= a$]

% = RichEdit.TextLength

Object:Ocx Object
x!: Single
a$: sexp

Description

RichEdit and TextBox control the value specifies the text appearing in the control.

ComboBox control (Style property set to 0 [Dropdown Combo] or to 1 [Simple Combo]) - returns or sets the text contained in the edit area. ComboBox control (Style property set to 2 [Dropdown List]) and ListBox control - returns the selected item in the list box; the value returned is always equivalent to the value returned by the expression List(ListIndex).

The TextLength is a RichEdit property returning the current number (Long) of characters in the control.

Example

```
OpenW 1 : AutoRedraw = 1
Ocx RichEdit red = "", 10, 10, 300, 400 :
  .MultiLine = True : .BorderStyle = 1
red.SelColor = 255 : red.SelText = "Hello " :
  red.SelColor = 0 : red.SelText = "World"
Text 320, 10, "Text Length: " & red.TextLength
Text 320, 30, "Plain Text:"
Ocx Label lbl = "", 320, 45, 300, 300 : .MultiLine
  = True : lbl.Text = red.Text
Do : Sleep : Until Me Is Nothing

Sub red_Change
  Text 320, 10, "Text Length: " & red.TextLength &
    Space(5)
  If Not lbl Is Nothing Then lbl.Text = red.Text
EndSub
```

See Also

ComboBox, ListBox, TextBox, RichEdit, Command, Option, CheckBox, Frame, Label, MenuItem, Node, Panel, ListItem, ColumnHeader, Tab

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

ToolTipText Property

Purpose

Specifies the text that appears as a ToolTip for an Ocx control. Available at design time and run time.

Syntax

Object.**ToolTipText** = *Txt*

Object:Ocx Object
Txt:sexp

Description

Specifies the text to use for the ToolTip. When the ToolTipText is assigned a value, a tooltip will be displayed when the mouse hovers over the object. The maximum number of characters you can specify for *Txt$* is 79.

Example

```
OpenW Center 1, , , 300, 120
Local Int32 x = (300 - (Screen.cxFrame * 2) - 120)
  / 2, y = (120 - (Screen.cyFrame * 2) -
  Screen.cyCaption - 22) / 2
Ocx Command cmd = "OK", x, y, 120, 22 :
  cmd.ToolTipText = "Press to close window"
Do : Sleep : Until Win_1 Is Nothing

Sub cmd_Click
  Win_1.Close
EndSub
```

Remarks

You can use this property to explain each object with a few words.

See Also

Form

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Visible Property

Purpose

Returns or sets a value indicating whether an object is visible or hidden.

Syntax

object.**Visible** [= *boolean*]

object:Ocx object

Description

To hide an object at startup, set the **Visible** property to **False** at design time. Setting this property in code enables you to hide and later redisplay a control at run time in response to a particular event.

Example

```
OpenW 1, 10, 10, 300, 200 : TitleW 1, "Window 1"
Ocx CheckBox chk = "Show Window 2", 10, 10, 120,
  14
Ocx Label lbl = "Window 2 is Invisible", 10, 30,
  120, 14
OpenW Hidden 2, 320, 10, 300, 200 : TitleW 2,
  "Window 2"
Do : Sleep : Until Win_1 Is Nothing Or Win_2 Is
  Nothing
CloseW 1 : CloseW 2

Sub chk_Click
```

```
  Win_2.Visible = -chk.Value
  lbl.Caption = "Window 2 is " & (Visible?
    (Win_2.hWnd) ? "Visible" : "Invisible")
EndSub
```

Remarks

Using the **Show** or **Hide** method on a form is the same as setting the form's **Visible** property in code to **True** or **False**, respectively.

See Also

Form

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

DoClick Method

Purpose

The **DoClick** method emulates a mouse click.

Syntax

object.**DoClick**

Object:Form, Command, Option, CheckBox, RichEdit, TextBox Ocx

Description

Emulates a mouse click in the named Ocx objects.

Example

```
Ocx Command cmd1 = "Click Me...", 10, 10, 100, 22
Ocx Command cmd2 = "...to Activate Me", 120, 10,
  100, 22
Global cmdclick As Byte
Do : Sleep : Until Me Is Nothing

Sub cmd1_Click
  cmdclick = 1 : cmd2.DoClick
EndSub

Sub cmd2_Click
  If cmdclick = 1 Then Message "Button 2 activated
    by Button 1"
  cmdclick = 0
EndSub
```

See Also

Form, Command, Option, CheckBox, RichEdit, TextBox

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

Move and Center Methods

Purpose

Moves a **Form** or Ocx control.

Syntax

*Object***.Move** [left] [, top] [, width] [, height]

*Object***.Center** [hWnd]

left, top, width, height:Single exp
hWnd:Handle exp

Description

For Forms the coordinate system is always in twips. Moving a form on the screen is always relative to the origin (0,0), which is the upper-left corner. When moving a control on a **Form** object (or an MDI child form on an MDI Form object), the coordinate system of the container object is used. The coordinate system or unit of measure is set with the **ScaleMode** property at design time. You can change the coordinate system at run time with the **Scale** method.

The **Center** method centers the form on the screen, or when specified in the center of another window *hWnd*. In the Form Editor a form can be centered by setting the **StartupMode** = 1.

Example

```
Form frm1 = "SDI", 20, 20, 300, 300
```

```
frm1_Load        ' Only LoadForm executes _Load
Do
  Sleep
Until Me Is Nothing

Sub cmd1_Click
  frm1.Center Screen.hWnd
EndSub

Sub frm1_Load
  ScaleMode = basPixels
  BackColor = col3DFace
  Ocx TreeView tv1
  .BackColor = frm1.BackColor
  Ocx ListView lvw1
  .BackColor = frm1.BackColor
  Ocx Command cmd1 = "Centre"
  frm1_ReSize
EndSub

Sub frm1_ReSize
  If IsNothing(tv1) Then Exit Sub
  tv1.Move 0, 0, ScaleWidth / 3, ScaleHeight
  lvw1.Move ScaleWidth / 3 , 40, ScaleWidth -
    ScaleWidth / 3, ScaleHeight
  cmd1.Move ScaleWidth / 3 + 10, 10, 100, 22
EndSub
```

Draws two Ocx controls inside a Form, a TreeView covering 1/3 of the client area and a ListView 2/3 with a command button towards the top of the screen. The ScaleMode is set to pixels. The Resize event sub is responsible for placing the controls using the current scaling mode and by clicking the command button, you can centre the form within the desktop.

Remarks

The coordinate system or unit of measure is set with the **ScaleMode** property at design time. You can change the coordinate system at run time with the **Scale** or **ScaleMode** method. For forms it is always twips.

Note on Center: Using *Form*.**Center** Screen.hWnd centres the object within the screen regardless of where the taskbar is; this is different to the **Center** parameter used with **OpenW** which centres a form within the area of the screen not covered by the taskbar. This is shown better by the example below:

```
OpenW 1, 10, 10, 160, 140
Ocx Command cmd = "Open centred window", 10, 10,
  125, 36 : cmd.WinStyle = cmd.WinStyle |
  BS_MULTILINE
Do : Sleep : Until Me Is Nothing

Sub cmd_Click
  Static act
  Select act
  Case 0 : OpenW Center 2, , , 100, 100 :
    cmd.Caption = "Open new window and Centre
    manually"
  Case 1 : OpenW 3, 0, 0, 100, 100 : Win_3.Center
    Screen.hWnd : cmd.Caption = "Close all windows"
  Case 2 : CloseW 3 : CloseW 2 : CloseW 1
  EndSelect
  Inc act
EndSub
```

See Also

Form, Left, Top, Width, Height

{Created by Sjouke Hamstra; Last updated: 19/10/2014 by James Gaite}

Refresh Method

Purpose

Forces a complete repaint of a form or control.

Syntax

Object.**Refresh**

Object:Ocx Object

Description

Generally, painting a form or control is handled automatically while no events are occurring. However, there may be situations where you want the form or control updated immediately.

Refresh invokes the *UpdateWindow* API function to send the object a WM_PAINT message if the window's update region is not empty. The function sends a WM_PAINT message directly to the window procedure of the specified window, bypassing the application queue. If the update region is empty, no message is sent.

Example

```
OpenW 1
Box 10, 10, 100, 100
Win_1.Refresh
```

Remarks

See Also

Form

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

SetFocus Method

Purpose

Moves the focus to the specified control or form.

Syntax

object.**SetFocus**

object:Ocx Object

Description

The object must be a **Form** object, or an Ocx control that can receive the focus. After invoking the **SetFocus** method, any user input is directed to the specified form or control.

You can only move the focus to a visible form or control. Because a form and controls on a form aren't visible until the form's Load event has finished, you can't use the **SetFocus** method to move the focus to the form being loaded in its own Load event unless you first use the **Show** method to show the form before the Form_Load event procedure is finished.

You also can't move the focus to a form or control if the **Enabled** property is set to **False**. If the **Enabled** property has been set to **False** at design time, you must first set it to **True** before it can receive the focus using the **SetFocus** method.

Example

```
Ocx TextBox tb = "", 10, 10, 140, 14 :
  .BorderStyle = 1 : .Text = "TextBox" : .ReadOnly
  = True
Ocx Command cmd = "Command Button", 160, 10, 140,
  22
Ocx Option opt(0) = "Give Focus to TextBox", 10,
  40, 180, 14 : opt(0).Value = 1
Ocx Option opt(1) = "Give Focus to Command
  Button", 10, 56, 180, 14
tb.SetFocus
Do : Sleep : Until Me Is Nothing

Sub opt_Click(Index%)
  If Index% = 0 : tb.SetFocus
  Else : cmd.SetFocus
  EndIf
EndSub
```

See Also

Form

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

ZOrder Method

Purpose

Places a specified **Form**, or Ocx control at the front or back of the z-order within its graphical level.

Syntax

object.**ZOrder** [*position = 0*]

object:Ocx Object
position:iexp

Description

The *position* parameter is optional, it indicates the position of object relative to other instances of the same object. If position is 0 or omitted, object is positioned at the front of the z-order. If position is 1, object is positioned at the back of the z-order.

Example

```
OpenW 1, 10, 10, 300, 200 : TitleW 1, Win_1.Name :
  Win_1.AutoRedraw = 1
Ocx Command cmd(1) = "Put to Back", 10, 10, 120,
  22
OpenW 2, 100, 100, 300, 200 : TitleW 2, Win_2.Name
  : Win_2.AutoRedraw = 1
Ocx Command cmd(2) = "Put to Back", 10, 10, 120,
  22
Do : Sleep  : Until Win_1 Is Nothing Or Win_2 Is
  Nothing
```

```
CloseW 1 : CloseW 2

Sub cmd_Click(Index%)
  If Index% = 1 Then Win_1.ZOrder 1
  If Index% = 2 Then Win_1.ZOrder 0
EndSub
```

Remarks

The z-order of objects can be set at design time by choosing the Bring To Front or Send To Back context menu commands. (Right click on an object)

See Also

Form, Arrange

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Click Event

Purpose

Occurs when the user presses and then releases a mouse button over an object. It can also occur when the value of a control is changed.

Syntax

Sub *object***_Click(**[*index* **As Integer**]**)**

object:Ocx Object
index:iexp

Description

For a **Form** object, this event occurs when the user clicks either a blank area or a disabled control. For a control, this event occurs when the user:

- Clicks a control with the left or right mouse button. With a **CheckBox**, **Command**, **ListBox**, or **Option** control, the Click event occurs only when the user clicks the left mouse button.

- Selects an item in a **ComboBox** or **ListBox** control, either by pressing the arrow keys or by clicking the mouse button.

- Presses the SPACEBAR when a **Command**, **Option**, or **CheckBox** control has the focus.

- Presses ENTER when a form has a **Command** control with its **Default** property set to **True**.

- Presses ESC when a form has a Cancel button - a **Command** control with its **Cancel** property set to **True**.

- Presses an access key for a control. For example, if the caption of a **Command** control is "&Go", pressing ALT+G triggers the event.

You can also trigger the Click event in code by:

- Setting a **Command** control's **Value** property to **True**.

- Setting an **Option** control's **Value** property to **True**.

- Changing a **CheckBox** control's **Value** property setting.

You can use a control's **Value** property to test the state of the control from code. Clicking a control generates MouseDown and MouseUp events in addition to the Click event. The order in which these three events occur varies from control to control. For example, for **ListBox** and **Command** controls, the events occur in this order: MouseDown, Click, and MouseUp. But for a **Label** control, the events occur in this order: MouseDown, MouseUp, and Click.

Example

```
Form frm1 = "(Dbl)Click Event", 20, 20, 300, 300
Ocx Command cmd(1) = "cmd1", 10, 10, 80, 24
Ocx Command cmd(2) = "cmd2", 10, 40, 80, 24
Ocx Command cmd(3) = "cmd3", 10, 70, 80, 24
Do
  Sleep
Until Me Is Nothing

Sub cmd_Click(Index%)
  Text 100, 30, "Click at " + Str(Index)
```

```
EndSub

Sub cmd_DblClick(Index%)
  Text 100, 30, "DblClick at " + Str(Index)
EndSub
```

Remarks

When you're attaching event procedures for these related events, be sure that their actions don't conflict. If the order of events is important in your application, test the control to determine the event order.

If there is code in the Click event, the DblClick event will never trigger, because the Click event is the first event to trigger between the two. As a result, the mouse click is intercepted by the Click event, so the DblClick event doesn't occur.

To distinguish between the left, right, and middle mouse buttons, use the MouseDown and MouseUp events.

See Also

Form, DblClick, MouseDown, MouseDblClick

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

DblClick Event

Purpose

Occurs when the user presses and releases a mouse button and then presses and releases it again over an object.

Syntax

Sub *object***_DblClick (**[*index* **As Integer**]**)**

object:Ocx Object
index:iexp

Description

For a form, the DblClick event occurs when the user double-clicks a disabled control or a blank area of a form. For a control, it occurs when the user:

- Double-clicks a control with the left mouse button.
- Double-clicks an item in a **ComboBox** control whose **Style** property is set to 1 (Simple) or in a **ListBox**.

The argument *Index* uniquely identifies a form or control if it's in a form or control array. You can use a DblClick event procedure for an implied action, such as double-clicking an icon to open a window or document. You can also use this type of procedure to carry out multiple steps with a single action, such as double-clicking to select an item in a list box and to close the dialog box.

Example

```
Form frm1 = "(Dbl)Click Event", 20, 20, 300, 300
Ocx Command cmd(1) = "cmd1", 10, 10, 80, 24
Ocx Command cmd(2) = "cmd2", 10, 40, 80, 24
Ocx Command cmd(3) = "cmd3", 10, 70, 80, 24
Do
  Sleep
Until Me Is Nothing

Sub cmd_Click(Index%)
  Text 100, 30, "Click at " + Str(Index)
EndSub

Sub cmd_DblClick(Index%)
  Text 100, 30, "DblClick at " + Str(Index)
EndSub
```

Remarks

For those objects that receive Mouse events, the events occur in this order: MouseDown, MouseUp, Click, DblClick, and MouseUp.

To distinguish between the left, right, and middle mouse buttons, use the MouseDblClick event.

See Also

Form, MouseDown, MouseDblClick, Click

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

GotFocus, LostFocus Events

Purpose

Occurs when an object receives or loses the focus respectively. The focus is shifted either by user action, such as tabbing to or clicking the object, or by changing the focus in code using the **SetFocus** method. A form receives the focus only when all visible controls are disabled or when it is explicitly given the focus.

Syntax

Sub *object***_GotFocus(**[*index%*]**)**

Sub *object***_LostFocus(**[*index%*]**)**

object:Ocx Object
index%iexp

Description

Typically, you use a **GotFocus** event procedure to specify the actions that occur when a control or form first receives the focus. For example, by attaching a GotFocus event procedure to each control on a form, you can guide the user by displaying brief instructions or status bar messages. You can also provide visual cues by enabling, disabling, or showing other controls that depend on the control that has the focus.

A **LostFocus** event procedure is primarily useful for verification and validation updates. Using **LostFocus** can cause validation to take place as the user moves the focus from the control. Another use for this type of event procedure is enabling, disabling, hiding, and displaying other objects as in a **GotFocus** event procedure. You can also reverse or change conditions that you set up in the object's **GotFocus** event procedure.

index% - An integer that uniquely identifies a form or control if it's in a form or control array.

Example

```
Form frm1 = "Key Events", 20, 20, 300, 300
Ocx TextBox txt(1) = , 10, 10, 150, 40
txt(1).BorderStyle = 1
Ocx TextBox txt(2) = , 10, 70, 150, 40
txt(2).BorderStyle = 1
Do
  Sleep
Until Me Is Nothing

Sub txt_GotFocus(Index%)
  ' Show focus with red.
  txt(Index).BackColor = RGB(255, 0, 0)
End Sub

Sub txt_LostFocus(Index%)
  ' Show loss of focus with blue.
  txt(Index).BackColor = RGB(0, 0, 255)
End Sub
```

Remarks

An object can receive the focus only if it's **Enabled** and **Visible** properties are set to **True**. To customize the keyboard interface in GFA-BASIC 32 for moving the focus, set the tab order by rearranging the controls using the 'Ocx

Overview' window or specify access keys for controls on a form.

See Also

Form, Activate, Enabled, SetFocus

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

KeyDown, KeyUp Events

Purpose

Occur when the user presses (KeyDown) or releases (KeyUp) a key while an object has the focus.

Syntax

Sub *object*__KeyDown(__[*index%*,] *Code&*, *Shift&*__)__

Sub *object*__KeyUp(__[*index%*,] *Code&*, *Shift&*__)__

object	*:Ocx Object*
index	*:iexp*
Code&, Shift&	*:Short exp*

Description

The KeyDown and KeyUp event syntaxes have these parts:

index% - An integer that uniquely identifies a control or form if it's in an array.

Code& - A key code, such as VK_F1 (the F1 key) or VK_HOME (the HOME key). To specify key codes, use the API VK_* constants. For a comprehensive list of key codes, see Key Codes and ASCII Values.

shift& - An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2).

Shift	1
Control	2
Alt	4

These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

KeyDown and **KeyUp** aren't invoked for:

- The ENTER key if the form has a **Command** control with the **Default** property set to **True**.

- The ESC key if the form has a **Command** control with the **Cancel** property set to **True**.

- The TAB key.

KeyDown and **KeyUp** interpret the uppercase and lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key) and *shift*, which indicates the state of *shift* + *key* and therefore returns either A or a.

Example

```
Form frm1 = "KeyDown, KeyUp Events", 20, 20, 300,
  300
Do
  Sleep
Until Me Is Nothing

Sub frm1_KeyDown(Code&, Shift&)
  Print "KeyDown - Code& = "; Code&
  If Shift& And 1
    Print "Shift pressed"
  Else If Shift& And 2
    Print "Ctrl pressed"
  Else If Shift& And 4
    Print "Alt pressed"
  EndIf
EndSub

Sub frm1_KeyUp(Code&, Shift&)
  Print "KeyUp - Code& = "; Code&
  If Shift& And 1
    Print "Shift pressed"
  Else If Shift& And 2
    Print "Ctrl pressed"
  Else If Shift& And 4
    Print "Alt pressed"
  EndIf
EndSub
```

Remarks

Use the **Screen_KeyPreview** event to create global keyboard-handling routines. This event sub receives these events before controls or the form receives the events.

Known Issues

According to previous documentation, by setting Code& to 0, the keypress would then be ignored; this does not happen.

To cancel the input from the keyboard, there are two methods available:

1. Use the **Screen_KeyPreview** event to catch the key combination early. Then, once you have processed it,

set *Cancel?* to True.

2. To cancel a simple keypress - let's say you want to disable the letter 'A' on the keyboard - use the older **Keypress** event as below:

```
OpenW 1 : Win_1.PrintWrap = True : FontName =
  "courier new"
Do : Sleep : Until Win_1 Is Nothing

Sub Win_1_KeyPress(Ascii&)
  If Chr(Ascii&) = "A" Or Chr(Ascii&) = "a"
    Ascii& = 0
  Else
    Print Chr(Ascii&);
  EndIf
EndSub
```

3. However, to intercept and cancel a more complex combination such as Ctrl-V (which is used to paste text in textboxes), a combination of **KeyDown** and **Keypress** is required as shown below.

```
Global diskey?, disv?
OpenW 1
Ocx TextBox tb = "", 10, 10, 200, 150 :
  .MultiLine = True : .BorderStyle = 1
Ocx Command cmd = "Disable Ctrl-V", 230, 10,
  100, 22
Clipboard.SetText "GFA Basic "
tb.SetFocus
Do : Sleep : Until Win_1 Is Nothing

Sub cmd_Click
  Local tbss As Int32 = tb.SelStart // Stores
    Textbox Caret position
  If disv? Then cmd.Caption = "Disable Ctrl-V"
  If Not disv? Then cmd.Caption = "Enable Ctrl-
    V"
  disv? = Not disv?
  tb.SetFocus                            // Returns
    focus to Textbox
  tb.SelStart = tbss                     // Replaces
    the caret where it was
EndSub

Sub tb_KeyDown(Code&, Shift&)
  Debug Code&
  If disv?                            // If Ctrl-V is
    disabled
    If Shift& = 2 And Code& = 86 // If Ctrl-V
      pressed
      diskey? = True                 // Disable
        Keypress
    EndIf
  EndIf
EndSub

Sub tb_KeyPress(Ascii&)
  If diskey?
    Ascii& = 0          // Cancels the keypress
    diskey? = False     // Resets the diskey?
      flag
  EndIf
EndSub
```

On occasions when using the example above with different 'shift key' combinations, the key press after the disqualified key combination may be ignored or not printed; if this happens, insert '`diskey? = False`' as the first line of the **KeyDown** sub.

See Also

Form, KeyPress, Screen_KeyPreview

{Created by Sjouke Hamstra; Last updated: 01/03/2017 by James Gaite}

KeyPress Event

Purpose

Occur when the user presses and releases an ANSI key.

Syntax

Sub *object***_KeyPress(**[*index%*,] *Ascii&***)**

object:Ocx Object
index:iexp
Ascii&Short exp

Description

The **KeyPress** event syntax has these parts:

*index%*An integer that uniquely identifies a control or form if it's in an array.

*Ascii&*An integer that returns a standard numeric ANSI keycode. Ascii& is passed by reference; changing it sends a different character to the object. For a full list of ASCII and ANSI (Windows 1252) values, see Key Codes and ASCII Values.

Changing *Ascii&* to 0 cancels the keystroke so the object receives no character.

Changing the value of the *Ascii&* argument changes the character displayed.

Example

```
Form frm1 = "Key Press", 20, 20, 300, 300 :
  .FontName = "courier new"
Do
  Sleep
Until Me Is Nothing

Sub frm1_KeyPress(Ascii&)
  // Converts any key pressed to upper case
  Local ch$ = Upper(Chr(Ascii&))
  Ascii& = Asc(ch$)
  Print ch$;
EndSub
```

Remarks

Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress doesn't indicate the physical state of the keyboard; instead, it passes a character.

Use the **Screen_KeyPreview** event to create global keyboard-handling routines. This event sub receives these events before controls or the form receives the events.

See Also

Form, KeyUp, KeyDown, Screen_KeyPreview

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

MouseDown, MouseUp Events

Purpose

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button.

Syntax

Sub Object**_MouseDown(**[index%,] *button&*, *shift&*, *x!*, *y!***)**

Sub Object**_MouseUp(**[index%,] *button&*, *shift&*, *x!*, *y!***)**

Object	*: Ocx Object*
button&, shift&	*: Short integer exp*
x!, y!	*: Single exp*

Description

ObjectReturns an Ocx object expression.

index%	Returns an integer that uniquely identifies a form or control if it's in a form or control array.
button&	Returns an integer that identifies the button that was pressed (MouseDown) or released (MouseUp) to cause the event. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of

	the bits is set, indicating the button that caused the event.
shift&	Returns an integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released. A bit is set if the key is down. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of shift would be 6.
x!, y!	Returns a number that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Use a **MouseDown** or **MouseUp** event procedure to specify actions that will occur when a given mouse button is pressed or released. Unlike the **Click** and **DblClick** events, **MouseDown** and **MouseUp** events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Ocx controls that are aligned at the border of the parent form (using **Align**) change the Scale settings of the parent. **ScaleLeft** and **ScaleTop** are set to the top-left pixel of the uncovered client area of the form. **ScaleWidth** and **ScaleHeight** are set to width and height of the uncovered area. The mouse coordinates returned from **MouseX**, **MouseY** and that are passed in the forms **MouseMove**, **MouseUp**, and **MouseDown** events are relative to the new origin.

Example

```
OpenW # 1 : FontName = "courier new"
Do
  Sleep
Until Win_1 Is Nothing

Sub Win_1_MouseDown(Button&, Shift&, x!, y!)
  Print AT(1, 1); "Mouse Down"
EndSub

Sub Win_1_MouseUp(Button&, Shift&, x!, y!)
  Print AT(1, 1); Space(10)
EndSub
```

Remarks

The following applies to both **Click** and **DblClick** events:

- If a mouse button is pressed while the pointer is over a form or control, that object "captures" the mouse and receives all mouse events up to and including the last **MouseUp** event. This implies that the x, y mouse-pointer coordinates returned by a mouse event may not always be in the internal area of the object that receives them.
- If mouse buttons are pressed in succession, the object that captures the mouse after the first press receives all mouse events until all buttons are released.

See Also

Form, Click, DblClick, MouseMove

{Created by Sjouke Hamstra; Last updated: 02/03/2018 by James Gaite}

MouseMove Event

Purpose

Occurs when the user moves the mouse.

Syntax

Sub *Object*_**MouseMove(**[index%,] *button&*, *shift&*, *x!*, *y!***)**

Object:Ocx Object
button&, shift&:Short integer exp
x!, y!:Single exp

Description

Object - Returns an Ocx object expression.

index% - Returns an integer that uniquely identifies a form or control if it's in a form or control array.

button - Returns an integer that identifies the button that was pressed (MouseDown) or released (MouseUp) to cause the event. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.

shift - Returns an integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released. A bit is set if the key is down. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of shift would be 6.

x, y - Returns a number that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

The **MouseMove** event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a **MouseMove** event whenever the mouse position is within its borders.

Ocx controls that are aligned at the border of the parent form (using **Align**) change the Scale settings of the parent. **ScaleLeft** and **ScaleTop** are set to the top-left pixel of the uncovered client area of the form. **ScaleWidth** and **ScaleHeight** are set to width and height of the uncovered area. The mouse coordinates returned from **MouseX**, **MouseY** and that are passed in the forms **MouseMove**, **MouseUp**, and **MouseDown** events are relative to the new origin.

Example

```
OpenW # 1 : FontName = "courier new"
// This next line replaces the mouse position co-
 ordinates ...
// ...printed in the default font when win_1 was
 opened and before...
```

```
// ...the fontname setting was enacted.
Win_1_MouseMove(0, 0, MouseX, MouseY)
Do
  Sleep
Until Me Is Nothing

Sub Win_1_MouseMove(Button&, Shift&, x!, y!)
  Print AT(1, 1); "Mouse Position: "; x!; " : ";
    y!; Space(10)
EndSub
```

Remarks

See Also

Form, Click, DblClick, MouseDown

{Created by Sjouke Hamstra; Last updated: 19/10/2014 by James Gaite}

Alignment Property

Purpose

Returns or sets a value that determines the text alignment in an object.

Syntax

object.**Alignment** [= *number*]

object:Label, TextBox, Panel Ocx Object
number:iexp

Description

For **Label, TextBox,** and **Panel** objects, the settings for number are:

Constant	Setting	Description
basLeftJustify	0	(Default) Text is left-aligned.
basRightJustify	1	Text is right-aligned.
basCenter	2	Text is centered.

Example

```
lbl1.Alignment = basCenter
```

See Also

Label, TextBox, Panel

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

BorderStyle Property

Purpose

Returns or sets the border style for an object.

Syntax

object.**BorderStyle** = [*value*]

object:Ocx Object
valueBorderStyleConst

Description

For a form, the **BorderStyle** property determines key characteristics that visually identify a form as either a general-purpose window or a dialog box. The value can be one of the following BorderStyleConst:

basNone	(0)	no border and no caption (Form only)
basFixedSingle	(1)	simple border (WS_BORDER)
basThick	(2)	double border (WS_THICKFRAME)

The form's **SizeAble** property affects the **BorderStyle** property and sets it to **basThick**.

Example

```
Ocx Label lbl = "No Border", 10, 10, 100, 16
Ocx Command cmd = "Add Single Border", 10, 35,
  100, 22
Do : Sleep : Until Me Is Nothing
```

```
Sub cmd_Click
  Static Byte cond = 0
  Inc cond : If cond = 3 Then cond = 0
  lbl.BorderStyle = cond
  Select cond
  Case 0 : cmd.Caption = "Add Single Border"
  Case 1 : cmd.Caption = "Add Double Border"
  Case 2 : cmd.Caption = "Remove Border"
  EndSelect
EndSub
```

Remarks

The **BorderStyle** is a property for all visible Ocx objects.

Known Issues

With **TextBoxes**, the positioning of the **BorderStyle** property is important. In the example below, clicking the command button will set the border to double width (incorrect behaviour).

```
Ocx TextBox tb = "", 10, 10, 300, 32 :
  tb.BorderStyle = 1 : tb.MultiLine = True
Ocx Command chk = "Set TextBox border to 1", 10,
  50, 150, 22
Do : Sleep  : Until Me Is Nothing

Sub chk_Click
  tb.BorderStyle = 1
EndSub
```

However, if you place the **BorderStyle** property change AFTER **Multiline**, then clicking the command button no longer doubles the width of the border.

See Also

Form, Appearance, SizeAble

{Created by Sjouke Hamstra; Last updated: 19/03/2015 by James Gaite}

MultiLine, MaxLength Properties

Purpose

MultiLine returns or sets a value indicating whether a **TextBox** or **RichEdit** control can accept and display multiple lines of text. **MaxLength** sets maximum number of character the control accepts.

Syntax

*object***.MultiLine** [**=** *boolean*]

*object***.MaxLength** [**=** *value*]

object:TextBox, RichEdit

Description

The default is 0, so that the edit control ignores carriage returns and restricts data to a single line.

A multiple-line **TextBox** control wraps text as the user types text extending beyond the text box. You can also add scroll bars to larger **TextBox** controls using the **ScrollBars** property. If no horizontal scroll bar is specified, the text in a multiple-line **TextBox** automatically wraps.

Maxlength specifies a long integer with the maximum number of characters a user can enter in the control. The default for the **MaxLength** property is 0, indicating no maximum other than that created by memory constraints on the user's system. Any number greater than 0 indicates the maximum number of characters.

Example

```
Ocx Label lbl = "Length of Text:", 10, 10,
  TextWidth("Length of Text:"), 14 : lbl.BackColor
  = $FFFFFF
Ocx TextBox tb1 = "", 15 + TextWidth("Length of
  Text:"), 10, 30, 14 : tb1.BorderStyle = 1  :
  tb1.MaxLength = 3
Ocx CheckBox chk = "Allow Multiline Text?", 10,
  30, 115, 14 : chk.BackColor = $FFFFFF
Ocx TextBox tb2 = "", 10, 50, 200, 60 :
  tb2.BorderStyle = 1
Do : Sleep : Until Me Is Nothing

Sub chk_Click
  // Changing the state of Multiline clears the
    text from the textbox
  Local t$ = tb2.Text (* Store the value in tb2 *)
  tb2.MultiLine = - chk.Value
  // It can, on occasion, change the Borderstyle
    setting too
  // Resetting using tb2.BorderStyle = 1 actually
    gives Borderstyle 2
  // although the BorderStyle property still
    returns a value of 1
  // This is known bug
  Trace tb2.BorderStyle
  tb2.BorderStyle = 1
  tb2.Text = t$
EndSub

Sub tb1_LostFocus
  If Not tb1 Is Nothing
    tb2.MaxLength = Val(tb1.Text)
  EndIf
EndSub
```

Remarks

On a form with a default button, pressing ENTER in a multiple-line **TextBox** control moves the focus to the next button and executes the button. To prevent this behaviour set **WantSpecial** = True or use the Ctrl-Enter key combination when entering the data.

See Also

TextBox, RichEdit

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Transparent Property

Purpose

Returns or sets the coloring mode for a control.

Syntax

object.**Transparent** [= Boolean]

object:Label, Frame, Image Ocx object

Description

The **Transparent** property enables coloring of the background when it is set to 0. When set to 1 the control is not filled.

Transparent does not generate a redraw.

Together with **BackColor** an **Image** control can be given a transparent color.

A **Frame** created in code has the **Transparent** property set to 1 (True), but in the Form Editor the default value is 0 (False).

Example

```
OpenW 1 : Win_1.BackColor = $00FFFF
Ocx Label lbl1 = "Non-transparent", 10, 10, 140,
  14
Ocx Label lbl2 = "Transparent", 10, 30, 140, 14 :
  lbl2.Transparent = True
Do : Sleep : Until Win_1 Is Nothing
```

Remarks

It is often necessary to invoke **ZOrder** to actually draw the **Transparent** control at the top of other controls.

See Also

Frame, Label, Image

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

HitTest Method

Purpose

Returns a value when the mouse is located at the coordinates of x and y.

Syntax

value = *Object*.**HitTest**(*x!*, *y!*)

value = *MonthView*.**HitTest(***x!*, *y!*, *Date* **As Date)**

Object:Label, ListView, TreeView
x!, y!:Single exp

Description

The coordinates should be in the Form's ScaleMode units.

Label - HitTest returns a Boolean = True if the mouse pointer is over a character pixel.

ListView - Returns a reference to a **ListItem** when the coordinate is over a ListView element. If no object exists at the specified coordinates, the **HitTest** method returns **Nothing**.

TreeView - Returns a reference to a **Node** object located at the coordinates x and y. If no object exists at the specified coordinates, the **HitTest** method returns **Nothing**.

MonthView - Returns a date located at the set of coordinates *x!, y!* in the ByRef variable *Date.* Most often used with drag-and-drop operations to determine if a drop target item is available at the present location. (*x!, y!*) are the coordinates of a target date in Twips and *Date* is the variable which receives the date under the mouse.

The **HitTest** method returns the following values which specify the part of the calendar over which the mouse pointer is hovering:

mvwCalendarBack(0) - The calendar background.

mvwCalendarDate(1) - Calendar date.

mvwCalendarDateNext(2) - When this area is clicked, the calendar displays the following month.

mvwCalendarDatePrev(3) - When this area is clicked, the calendar displays the previous month.

mvwCalendarDay(4) - The day labels above the dates.

mvwCalendarWeekNum(5) - The week number, if ShowWeekNumbers is set to True.

mvwNoWhere(6) - Bottom edge of the calendar.

mvwTitleBack(7) - Background of the calendar.

mvwTitleBtnNext(8) - The Next button in the title area.

mvwTitleBtnPrev(9) - The Previous button in the title area.

mvwTitleMonth(10) - The month string in the title.

mvwTitleYear(11) - The year string in the title.

mvwTodayLink(12) - When this area is clicked, the calendar displays the current month and day. Only available if **ShowToday** is set to True.

Example

An example with ListView

```
OpenW 1 : Set Me = Win_1
Global a$, m As Int, n As Int
Dim li As ListItem
Ocx ListView lv1 = , 10, 10, 500, 150 : lv1.View =
  3
For n = 1 To 5 : lv1.ColumnHeaders.Add , ,
  "Column" & n : Next n
For n = 1 To 5 :
  a$ = "" : For m = 1 To 5 : a$ = a$ & "Item " &
    ((n - 1) * 5) + m & Iif(m <> 5, ";", "") : Next
    m
  lv1.Add , , "" : lv1(n).AllText = a$ : If n = 2
    Then lv1(n).Ghosted = True
Next n
lv1.FullRowSelect = True // If this is omitted
  then HitTest only works on the first column
Ocx Label res = "", 10, 200, 150, 15  :
  res.BackColor = RGB(255, 255, 255)
Do : Sleep : Until Me Is Nothing

Sub lv1_MouseMove(Button&, Shift&, x!, y!)
  // This is not called if you are hovering over a
    column header
  x! = TwipsToPixelX(x!) : y! = TwipsToPixelY(y!)
  If lv1.HitTest(x!, y!) Is Nothing
    res.Caption = ""
  Else
    Set li = lv1.HitTest(x!, y!)
    res.Caption = "Result: Line" & li.Index
  EndIf
EndSub
```

Known Issues

The **HitTest** method does not appear to work with labels. There is a workaround (listed below), although it only works for single line labels:

```
OpenW 1
Ocx Label lbl = "A plain old label", 10, 10, 150,
  15
Ocx Label res = "", 10, 30, 150, 15 :
  res.BackColor = RGB(255, 255, 255)
Do : Sleep : Until Me Is Nothing

Sub lbl_MouseMove(Button&, Shift&, x!, y!)
  x! = TwipsToPixelX(x!) : y! = TwipsToPixelY(y!)
  If x! < TwipsToPixelX(lbl.TextWidth(lbl.Text))
    And y! <
    TwipsToPixelY(lbl.TextHeight(lbl.Text))
    // instead of: If lbl.HitTest(x!, y!)
    res.Caption = "Hovering over label text"
  Else
    res.Caption = ""
  EndIf
EndSub

Sub Win_1_MouseMove(Button&, Shift&, x!, y!)
  If Not res Is Nothing Then res.Caption = ""
EndSub
```

A crude option for labels with more than one line is to check that the pixel under the mousepointer is not the BackColor of the label. In this case, the *lbl_MouseMove* sub-routine would look like this:

```
Sub lbl_MouseMove(Button&, Shift&, x!, y!)
  x! = TwipsToPixelX(x!) : y! = TwipsToPixelY(y!)
  Local hdc As Long = GetWindowDC(lbl.hWnd), bcol
    As Int32 = lbl.BackColor, col As Int32 =
    GetPixel(hdc, x!, y!)
  // Check to see if bcol is a system colour and,
    if so, convert
  If (bcol And $FF000000) = $80000000 Then bcol =
    SysCol(bcol And $FF)
  If col <> bcol  // instead of: If lbl.HitTest(x!,
    y!)
    res.Caption = "Hovering over label text"
  Else
    res.Caption = ""
  EndIf
EndSub
```

See Also

Label, ListView, ListItem, TreeView, Node

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

AutoSize, Stretch, Tile Properties

Purpose

Control how a graphic is displayed in an **Image** control.

Syntax

*Image***.AutoSize** [**=** *Boolean*]

*Image***.Stretch** [**=** *Boolean*]

*Image***.Tile** [**=** *Boolean*]

Description

AutoSize = True	The **Image** control resizes to fit the image.
AutoSize = False	(Default) The **Image** control does not resize.
Stretch = True	The graphic resizes to fit the control. Resizing the control also resizes the graphic it contains.
Stretch = False	(Default) The graphic keeps it original size.
Tile = True	The graphic is tiled across the Image control.
Tile = False	(Default) The graphic keeps it original size and position.

Remarks

A **Form** OCX control can also be used to display a picture. A **Form** OCX is the GFA-BASIC 32 implementation of a VB PictureBox.

See Also

Image, Form

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

TabStripName and TabStripIndex properties

Purpose

The **TabStripIndex** property returns whether the **Frame** is part of a **TabStrip** Ocx.

Syntax

% = object. **TabStripIndex**

Tab.**TabStripName**

object:Frame, Form, Image

Description

The **TabStripIndex** property returns the index of the **Tab** if it is owned by a **TabStrip** Ocx. If it is not part of a **TabStrip** it returns 0.

TabStripName returns a string containing the Ocx name of the **TabStrip** parent the **Tab** belongs to.

Example

```
Form Hidden Center frm1 = "TabStrip", , , 400, 300
Ocx TabStrip tbs = , 20, 20, ScaleWidth - 40,
  ScaleHeight - 40
Ocx Frame fr1 = "Tab #1"
Ocx Frame fr2 = "Tab #2"
Ocx Frame fr3 = "Tab #3"
Ocx Frame fr4 = "Tab #4"
```

```
OcxOcx fr1 Option opt1 = "Option #1", 20, 20, 80,
  24
OcxOcx fr1 Option opt2 = "Option #2", 20, 50, 80,
  24
OcxOcx fr2 CheckBox chk1 = "Check #1", 20, 20, 80,
  24
OcxOcx fr2 CheckBox chk2 = "Check #2", 20, 50, 80,
  24
OcxOcx fr3 TextBox txt1 = "TextBox #1", 20, 20,
  280, 40
OcxOcx fr3 TextBox txt2 = "TextBox #2", 20, 130,
  280, 40
OcxOcx fr4 Command cmd1 = "Command #1", 90, 20,
  80, 24
OcxOcx fr4 Command cmd2 = "Command #2", 90, 50,
  80, 24
tbs.Tabs.Add 1, , fr1.Caption , , fr1
tbs.AddItem 2, , fr2.Caption, , fr2
tbs.Add 3, , fr3.Caption, , fr3
tbs.AddItem 4, , fr4.Caption , , fr4
frm1.Show
Text 0, 0, "TabStripIndex: " & fr2.TabStripIndex &
  " name: " & tbs(2).TabStripName
frm1.Refresh
tbs(2).Selected = True
Do
  Sleep
Until Me Is Nothing

Sub tbs_Change
  Local tsi As Int32
  Switch tbs.SelectedIndex
  Case 1 : opt1.SetFocus : tsi = fr1.TabStripIndex
  Case 2 : chk1.SetFocus : tsi = fr2.TabStripIndex
  Case 3 : txt1.SetFocus : tsi = fr3.TabStripIndex
  Case 4 : cmd1.SetFocus : tsi = fr4.TabStripIndex
  EndSwitch
  Text 0, 0, "TabStripIndex: " & tsi & " Name: " &
    tbs(tbs.SelectedIndex).TabStripName
End Sub
```

See Also

TabStrip, Frame, Form, Image

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

HideSelection Property

Purpose

Returns a value that determines whether selected text appears highlighted when a control loses the focus.

Syntax

object.**HideSelection** [= True | False]

object:Textbox, RichEdit, ListView Ocx

Description

Normally, an edit control hides the selection when the control loses the input focus and inverts the selection when the control receives the input focus. Specifying **HideSelection** = 0 deletes this default action.

Example

```
Ocx TextBox txt1 = "", 10, 10, 100, 14 :
  txt1.BorderStyle = 1 : txt1.HideSelection = 0 :
  txt1 = "TextBox 1"
Ocx TextBox txt2 = "", 10, 30, 100, 14 :
  txt2.BorderStyle = 1 : txt2 = "TextBox 2"
Do : Sleep : Until Me Is Nothing
```

See Also

TextBox, RichEdit, ListView

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

PassWordChar Property

Purpose

Returns or sets a character to be used as a password character.

Syntax

object.**PassWordChar** [= chr$]

object:Textbox, RichEdit Ocx
chr$:sexp

Description

Displays all characters as an asterisk (*****) as they are typed into the edit control. An application can use the **PassWordChar** property to change the character that is displayed.

Setting **PassWordChar** = "" disables the password mode.

Example

```
Global tpos As Int32
Text 10, 10, "Password: "
Ocx TextBox tb = "", 65, 10, 100, 14 :
  tb.BorderStyle = 1 : tb.PassWordChar = "*"
Ocx Command cmd = "Show", 170, 8, 60, 20
Ocx CheckBox chk = "Use '#' character instead of '
  * '", 10, 40, 175, 14
tb.SetFocus
Do : Sleep : Until Me Is Nothing
```

```
Sub chk_Click
  Select chk.Value
  Case 0 : tb.PassWordChar = "*"
  Case 1 : tb.PassWordChar = "#"
  EndSelect
  tb.SetFocus
EndSub

Sub cmd_MouseDown(Button&, Shift&, x!, y!)
  tb.PassWordChar = ""
EndSub

Sub cmd_MouseUp(Button&, Shift&, x!, y!)
  chk_Click
EndSub

Sub tb_GotFocus
  tb.SelStart = tpos
EndSub

Sub tb_LostFocus
  If Not tb Is Nothing Then tpos = tb.SelStart
EndSub
```

Known Issues

*tb.***PassWordChar** works like *tb.***PassWordChar**="" rather than *tb.***PassWordChar**="*" as it should.

See Also

TextBox, RichEdit

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

ReadOnly, Locked, SelProtected Property, Protected event

Purpose

Returns or sets the read-only style of (part of) the text of an edit control.

Syntax

Object.**ReadOnly** [= *Boolean*]

*RichEdit***.Locked** [= *boolean*]

*RichEdit***.SelProtected** [= *variant*]

Sub RichEdit_**Protected**(Start&, End&, Cancel?)

Object: TextBox, RichEdit Ocx

Description

ReadOnly returns or sets the read-only style (ES_READONLY) of an edit control. With this style you cannot change the text within the edit control.

Locked returns or sets a value indicating whether the contents in a **RichEdit** control can be edited. You can scroll and highlight the text in the control, but you can't edit it. The program can still modify the text by changing the **Text** property.

SelProtected returns or sets a value which determines if the current selection is protected. Protected text looks the same a regular text, but cannot be modified by the end-user. That is, the text cannot be changed during run time. This allows you to create forms with the **RichRdit** control, and have areas that cannot be modified by the end user. **SelProtected** can return **Null** meaning that the selection contains a mix of protected and non-protected characters. It can be assigned a Boolean, meaning (True) that all the characters in the selection are protected, or (False) none of the characters in the selection are protected.

The **Protected** event notifies that the user is taking an action that would change a protected range of text. The Cancel? parameter provides the event the means to allow or prevent the change. Set Cancel? = False to accept the change.

Example

```
Local n As Int32
Ocx RichEdit rtb = "", 10, 10, 200, 300 :
  rtb.MultiLine = True : rtb.BorderStyle = 1 :
  rtb.ScrollBars = 2
For n = 1 To 100 : rtb.Text = rtb.Text & "This is
  a rich text edit box" & #13#10  : Next n
Ocx CheckBox chk(1) = "Locked", 230, 10, 100, 14
Ocx CheckBox chk(2) = "Read Only", 230, 30, 100,
  14
Ocx Command cmd = "Protect Selection", 230, 50,
  100, 22 : cmd.Enabled = False
Do : Sleep : Until Me Is Nothing

Sub chk_Click(Index%)
  rtb.Locked = -chk(1).Value
  rtb.ReadOnly = -chk(2).Value
```

```
EndSub

Sub cmd_Click
  rtb.SelProtected = True
EndSub

Sub rtb_MouseUp(Button&, Shift&, x!, y!)
  If Not cmd Is Nothing
    cmd.Enabled = (rtb.SelLength = 0 ? False :
      True)
  EndIf
EndSub

Sub rtb_Protected(Start%, End%, Cancel?)
  If MsgBox("An attempt was made to edit or access
    Protected text"#13#10#13#10 & _
    "Do you wish to continue with the deletion?",
      MB_YESNO, "Confirm Delete") = IDNO
    Cancel? = True
  EndIf
EndSub
```

See Also

TextBox, RichEdit

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

ScrollBars Property

Purpose

Returns or sets a value indicating whether an object has horizontal or vertical scroll bars.

Syntax

object.**ScrollBars** [= value]

object:Form, TextBox object
value:iexp

Description

The following values are allowed:

basNoScroll (0)	(Default) None
basHorizontal (1)	Horizontal
basVertical (2)	Vertical
basBoth (3)	Both scrollbars

For a **TextBox** control with setting 1 (Horizontal), 2 (Vertical), or 3 (Both), you must set the **MultiLine** property to **True**.

Example

```
OpenW 1
Ocx CheckBox chk(0) = "Show Horizontal Scroll
 Bar", 10, 10, 200, 14
```

```
Ocx CheckBox chk(1) = "Show Vertical Scroll Bar",
  10, 30, 200, 14
Do : Sleep : Until Win_1 Is Nothing

Sub chk_Click(Index%)
  If chk(Index%).Value = 0
    Win_1.ScrollBars = Bclr(Win_1.ScrollBars,
      Index%)
  Else
    Win_1.ScrollBars = Bset(Win_1.ScrollBars,
      Index%)
  EndIf
EndSub
```

Remarks

The scrollbar minimum, maximum, step size, and current position can be set with **HSc*** and **VSc*** properties.

The scrolling events are handled with **HScroll**, **HScrolling**, **VScroll**, and **VScrolling** event subs.

Known Issue

If you wish the scroll bar(s) to be visible only when your work area grows bigger than the window area, toggling the **ScrollBar** property between 0, 1, 2 and 3 has been known to cause a fatal error. Instead, it is advised to use the **HScMax** and **VScMax** properties to achieve the same end as setting these properties to zero causes the respective scroll bar to disappear. It is also possible to use the *EnableScrollBar()* API to disable the scroll bars but keep them visible. The example below illustrates how this works:

```
OpenW 1 : Win_1.ScrollBars = 3 : Win_1.BackColor =
  $8000000f
```

```
Ocx Option opt(1) = "Enable Horizontal Scrollbar",
  10, 10, 200, 15 : opt(1).Value = 1
Ocx Option opt(2) = "Disable Horizontal
  Scrollbar", 10, 25, 200, 15
Ocx Command cmd // Breaks option groups
Ocx Option opt(3) = "Show Horizontal Scrollbar",
  10, 50, 200, 15 : opt(3).Value = 1
Ocx Option opt(4) = "Hide Horizontal Scrollbar",
  10, 65, 200, 15
Do : Sleep : Until Win_1 Is Nothing

Sub opt_Click(Index%)
  Select Index%
  Case 1 : ~EnableScrollBar(Win_1.hWnd, SB_HORZ,
    ESB_ENABLE_BOTH)
  Case 2 : ~EnableScrollBar(Win_1.hWnd, SB_HORZ,
    ESB_DISABLE_BOTH)
  Case 3 : Win_1.HScMax = 1000
  Case 4 : Win_1.HScMax = 0
  EndSelect
EndSub
```

The same can be done for the vertical scroll bar by substituting SB_VERT and **VScMax** for SB_HORZ and **HScMax** respectively.

See Also

Form, TextBox, HScroll, HScrolling, VScroll, VScrolling, VScMax, VScMin, VScPos, VScPage, VScStep, VScTrack, HScMax, HScMin, HScPos, HScPage, HScStep, HScTrack

{Created by Sjouke Hamstra; Last updated: 02/07/2015 by James Gaite}

SelLength, SelStart, SelText Properties

Purpose

SelLength returns or sets the number of characters selected. **SelStart** returns or sets the starting point of text selected; indicates the position of the insertion point if no text is selected. **SelText** and **SelRTF** return or set the string containing the currently selected text.

Syntax

*object***.SelLength** [**=** *number*]

*object***.SelStart** [**=** *index*]

*object***.SelText** [**=** *string*]

object:TextBox,RichEdit

Description

SelLength = *number* specifies the number of characters selected. For **SelLength** and **SelStart**, the valid range of settings is 0 to text length - the total number of characters in the edit area of a **TextBox** control.

SelStart = *index* specifies the starting point of the selected text.

Use these properties for tasks such as setting the insertion point, establishing an insertion range, selecting substrings in a control, or clearing text. Used in conjunction with the

Clipboard object, these properties are useful for copy, cut, and paste operations.

Setting **SelStart** greater than the text length sets the property to the existing text length; changing **SelStart** changes the selection to an insertion point and sets **SelLength** to 0.

SelText returns or sets plain text. Setting new selected text sets **SelLength** to 0 and replaces the selected text with the new string.

Example

```
Ocx TextBox TBox1 = "", 100, 10, 100, 24
TBox1.BorderStyle = 1
TBox1 = "This is a Test"
TBox1.FontBold = 0
TBox1.BackColor = RGB(224, 224, 224)
TBox1.ForeColor = RGB(255, 0, 0)
TBox1.SelStart = 3
TBox1.SelLength = 4
TBox1.SetFocus
Do : Sleep : Until Me Is Nothing
```

See Also

TextBox, RichEdit, Ocx

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

WantSpecial Property

Purpose

Changes the actions of the ENTER and/or TAB keys within a multiline **TextBox** or **RichEdit** control.

Syntax

object.**WantSpecial** [= value]

object:Textbox, RichEdit Ocx

Description

WantSpecial = 1 specifies that a Carriage Return and Line Feed (CRLF) be inserted when the user presses the ENTER key while entering text into a multiple-line edit control in a form rather than activating the form's default pushbutton. Note that if there is no default command button, then ENTER automatically enters a CRLF within the object; similarly, the use of Ctrl+ENTER always enters a CRLF within the object, regardless of whether there is a default command button or not. This style has no effect on a single-line edit control.

WantSpecial = 2 prevents the TAB key moving the focus to the next control in a **TextBox** or **RichEdit** control; this setting also converts any Shift+TAB key combination, which would normally move the focus to the previous control, into a tab. Note that the Ctrl+TAB key combination always inserts a tab regardless of this setting.

WantSpecial = 3 enables both options (TAB & ENTER).

Example

```
Ocx TextBox txt = "", 10, 10, 200, 80 : .MultiLine
  = True : .BorderStyle = 1
Ocx CheckBox chk(1) = "Allow TAB within TextBox",
  220, 10, 160, 14 : chk(1).TabStop = False
Ocx CheckBox chk(0) = "Restrict ENTER to TextBox",
  220, 25, 160, 14 : chk(0).TabStop = False
Ocx Command cmd = "Close", 60, 100, 100, 22 :
  cmd.Default = True
Do : Sleep : Until Me Is Nothing

Sub chk_Click(Index%)
  Local Int ws = txt.WantSpecial
  Bchg ws, Index%
  txt.WantSpecial = ws
EndSub

Sub cmd_Click
  Me.Close
EndSub
```

See Also

TextBox, RichEdit

{Created by Sjouke Hamstra; Last updated: 29/06/2015 by James Gaite}

LineCount, LineFromChar, CharFromLine, RowFromChar, ColFromChar, GetLineFromChar Methods

Purpose

These **TextBox** and **RichEdit** control methods return information about positions in the text.

Syntax

% = *Object*.**LineCount** (or RichEdit.**LineCnt**)

% = *Object*.**LineFromChar**(index%)

% = *Object*.**CharFromLine**(index%)

% = *Object*.**RowFromChar**(index%)

% = *Object*.**ColFromChar**(index%)

% = *Object*.**GetLineFromChar**(index%)

Object:TextBox, RichEdit Ocx

Description

LineCount (property of both objects) and **LineCnt** (RichEdit only) retrieve the number of lines in a multiline edit control. If the edit control is empty, the return value is 1.

LineFromChar() retrieves the index of the line that contains the specified character index in a multiline edit control. **CharFromLine**() retrieves the character index of a line in a multiline edit control. The character index is the number of characters from the beginning of the edit control to the specified line.

RowFromChar() retrieves the y-coordinate of the specified character in an edit control. **ColFromChar**() retrieves the x-coordinate of the specified character in an edit control. The coordinates are relative to the left-top corner of the control.

GetLineFromChar()retrieves the index of the line that contains the specified character index in a multiline edit control. Same as **LineFromChar**().

Example

```
Global n As Int32
AutoRedraw = 1
Ocx TextBox tb = "", 10, 10, 210, 200 : .MultiLine
  = True : .BorderStyle = 1 : .ScrollBars = 2
For n = 1 To 100 : tb.Text = tb.Text & "Box" & n &
  ", " : Next n
Ocx Command cmd = "Add another box", 230, 10, 100,
  22
tb_Stats
Do : Sleep : Until Me Is Nothing

Sub cmd_Click
  Local tbss = tb.SelStart
  tb.Text = tb.Text & "Box" & n & "," : Inc n
  tb.SetFocus : tb.SelStart = tbss
  tb_Stats
EndSub
```

```
Sub tb_KeyUp(Code&, Shift&)
  tb_Stats
EndSub

Sub tb_MouseUp(Button&, Shift&, x!, y!)
  tb_Stats
EndSub

Sub tb_Stats
  Local tl = tb.LineFromChar(tb.SelStart)
  Text 230, 40, "Number of Lines:" & tb.LineCount &
    "     "
  Text 230, 56, "Line Position of Caret:" &
    tb.LineFromChar(tb.SelStart) & "     "
  Text 230, 72, "Character Position of Caret:" &
    tb.SelStart & "     "
  Text 230, 88, "Character No at Start of Caret
    Line:" & tb.CharFromLine(tl) & "     "
  Text 230, 104, "X Position of Caret:" &
    tb.ColFromChar(tb.SelStart) & "     "
  Text 230, 120, "Y Position of Caret:" &
    tb.RowFromChar(tb.SelStart) & "     "
EndSub
```

See Also

TextBox, RichEdit

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Scroll, ScrollCaret Methods

Purpose

Scrolls the text in a **TextBox** vertically or horizontally.

Syntax

object.**Scroll**(x , y)

object.**ScrollCaret**()

object:TextBox, RichEdit

Description

Scroll(x, y) scrolls the text vertically or horizontally in a multi-line edit control where:

*x*Specifies the number of characters to scroll horizontally.

y Specifies the number of lines to scroll vertically.

Scroll(x, y) works like a function and returns an Empty value; similar to in-built APIs, **Scroll** must be prefaced with **~** or **Void**, otherwise the 'Something Missing' error will be raised.

ScrollCaret scrolls the caret into view in the control.

Example

```
OpenW 1
Ocx TextBox tb = "", 10, 10, 300, 300 : .MultiLine
  = True : .ScrollBars = 2 : .BorderStyle = 1
```

```
Global Int32 n, tbpos
For n = 0 To Screen.FontCount - 1
  tb.Text = tb.Text & Screen.Fonts(n) & ", "
Next n
Ocx Command cmd(1) = "Scroll Up", 320, 10, 100, 22
Ocx Command cmd(3) = "Scroll Down", 320, 40, 100,
  22
Ocx Command cmd(4) = "Scroll to Caret", 320, 70,
  100, 22
Do : Sleep : Until Win_1 Is Nothing

Sub cmd_Click(Index%)
  tb.SetFocus
  If Index% = 4 Then tb.ScrollCaret : Exit Sub
  ~tb.Scroll(0, (Index% - 2) * 3)
EndSub

Sub tb_GotFocus
  tb.SelStart = tbpos
EndSub

Sub tb_LostFocus
  If Not tb Is Nothing Then tbpos = tb.SelStart
EndSub
```

See Also

TextBox, RichEdit

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Change, SelChange Event

Purpose

Indicates the contents of the current selection of a **TextBox** or **RichEdit** control have changed.

Syntax

Sub *object*_**Change(**[*index%*]**)**

Sub *object*_**SelChange(**[*index%*]**)**

object:TextBox, RichEdit Ocx
index:iexp (identifies a form or control if it's in a form or control array)

Description

The **Change** event procedure can synchronize or coordinate data display among controls. For example, you can use a **Change** event procedure to update the contents of another control. Or you can use a Change event procedure to display data and formulas in a work area and results in another area.

You can use the SelChange event to check the various properties that give information about the current selection (such as **SelBold**) so you can update buttons in a toolbar, for example.

Example

```
OpenW 1
```

```
Ocx TextBox TBox1 = "", 100, 10, 100, 24
TBox1.BorderStyle = 1
TBox1 = "This is a Test"
TBox1.FontBold = 0
TBox1.BackColor = RGB(224, 224, 224)
TBox1.ForeColor = RGB(255, 0, 0)
TBox1.SelStart = 3
TBox1.SelLength = 4
Do
  Sleep
Loop Until Me Is Nothing

Sub TBox1_Change
  Print "Change "; TBox1
EndSub

Sub TBox1_SelChange
  Print "SelChange "; TBox1.SelText
EndSub
```

Remarks

A Change event procedure can sometimes cause a cascading event. This occurs when the control's Change event alters the control's contents, for example, by setting a property in code that determines the control's value, such as the **Text** property setting for a **TextBox** control. To prevent a cascading event avoid creating two or more controls whose Change event procedures affect each other, for example, two **TextBox** controls that update each other during their **Change** events.

See Also

TextBox, RichEdit

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

SelAlignment, SelBullet, BulletIndent Property

Purpose

SelAlignment returns or sets a value that controls the alignment of the paragraphs in a **RichEdit** control.

SelBullet returns or sets a value that determines if a paragraph in the **RichEdit** control containing the current selection or insertion point has the bullet style. **BulletIndent** returns or sets the amount of indent used when **SelBullet** is set to **True**.

Syntax

object.**SelAlignment** [= *variant*]

object.**SelBullet** [= *variant*]

object.**BulletIndent** [= *variant*]

object:RichEdit

Description

The **SelAlignment** property determines paragraph alignment for all paragraphs that have text in the current selection or for the paragraph containing the insertion point if no text is selected. SelAlignment can be set to **basLeftJustify** (0), **basRightJustify** (1), and **basCenter** (2).

Use the **SelBullet** property to build a list of bulleted items in a **RichEdit** control. **SelBullet** returns and sets a **Variant** (Long) that determines the bullet style of the paragraph(s). The value is True when the paragraphs in the selection have the bullet style, it is False when not.

The **BulletIndent** property determines the amount of indent when **SelBullet** = True. Note, though, that the bullet point does not move with **SelBullet**, just the text following the bullet point; the bullet point itself is controlled by **SelIndent** - this is the intended behaviour and not a bug.

These properties returns **Null** if the selection spans more than one paragraph with different alignments or contains a mixture of bullet and non-bullet styles

Example

```
OpenW 1 : AutoRedraw = 1
Global Int32 n, rdpos
Ocx RichEdit red = "", 10, 10, 200, 200 :
  .MultiLine = True : .BorderStyle = 1 :
  .BulletIndent = 100
Ocx CheckBox chk(0) = "Bullet Points on", 230, 10,
  140, 14
Ocx TextBox tb = "", 230, 30, 60, 14 :
  .BorderStyle = 1 : .ReadOnly = True : Text 297,
  31, "Bullet Indent"
Ocx UpDown up : .BuddyControl = tb : .Increment =
  200 : .Min = 0 : .Max = 1000 : .Value =
  red.BulletIndent
Text 230, 54, "Alignment:"
Ocx ComboBox cmb = "", 280, 50, 100, 14 : .Style =
  2
cmb.AddItem "Left", 0 : cmb.AddItem "Centre", 2 :
  cmb.AddItem "Right", 1
For n = 0 To 2
```

```
  If red.SelAlignment = cmb.ItemData(n) Then
    cmb.ListIndex = n
Next n
Do : Sleep : Until Me Is Nothing

Sub chk_Click(Index%)
  red.SelBullet = chk(0).Value
  red.SetFocus
EndSub

Sub cmb_Click
  red.SelAlignment = cmb.ItemData(cmb.ListIndex)
  red.SetFocus
EndSub

Sub red_Change
  If Not IsNothing(chk(0)) Then chk(0).Value =
    red.SelBullet
EndSub

Sub red_GotFocus
  red.SelStart = rdpos
EndSub

Sub red_LostFocus
  If Not red Is Nothing Then rdpos = red.SelStart
EndSub

Sub up_Change
  red.SetFocus
  ' red.BulletIndent = up.Value
  // BulletIndent just moves the text but not the
    bullet point
  red.SelIndent = up.Value
  // SelIndent moved both the bullet point and
    text.
EndSub
```

To find if the current selection contains some (but not all) bulleted text, use the following code:

```
If IsNull(RichEdit1.SelBullet) = True
  ' Code selection has mixed style.
ElseIf RichEdit1.SelBullet = True
  RichEdit1.BulletIndent = 1000
End If
```

Remarks

Null differs from zero, these properties can only be queried with **IsNull**().

See Also

RichEdit, SelHangingIndent, SelIndent, SelRightIndent, IsNull

{Created by Sjouke Hamstra; Last updated: 18/12/2015 by James Gaite}

CharFormat, DefCharFormat, ParaFormat Property

Purpose

Return or set formatting for a **RichEdit** control.

Syntax

object.**CharFormat** [= format$]

object.**DefCharFormat** [= format$]

object.**ParaFormat** [= format$]

object:RichEdit

Description

The *format*$ value for **CharFormat** and **DefCharFormat** contains the setting for the character formatting. Each attribute is identified with a character and when necessary followed by a value.

The **DefCharFormat** property is used to set and retrieve the *default character formatting,* which is the formatting applied to any subsequently inserted characters. For example, if an application sets the default character formatting to bold and the user then types a character, that character is bold. Initially, **DefCharFormat** returns:

"biuspC0Y165O0T0F0'MS Sans Serif"

A capital enables the attribute, a lowercase character disables it. The following settings can be used:

"B/b" - Bold; "I/i" - Italic; "U/u" - UnderLine; "S/s" - Strikeout; "P/p" - Protected; "C" & Dec$(RGB()) - Color; "Y" & Dec(t) - Character height in twips; "O" & Dec(t) - Character offset, in twips, from the baseline; "T" - CharSet; "F" - PitchAndFamily; """&fontname - Font face name.

format$ = **ParaFormat** returns the current paragraph formatting for the selected text. This property is used to specify paragraph formatting attributes. The default value is "A0N0O0+0R0S0", meaning.

"A" - Alignment; N - Numbering; "O" - Offset; "+" - relative indenting; "R" - RightIndent; "S" - StartIndent ; "T" - Tabstops. All attributes are followed by a decimal value specifying the attributes setting in twips.

Example

```
Ocx RichEdit rtf = "", 10, 10, 400, 150 :
  rtf.BorderStyle = 1 : rtf.SetFocus
Ocx Command cmd(1) = "18pt Text (Italic)", 10,
  170, 80, 22
Ocx Command cmd(2) = "36pt Text (Bold)", 100, 170,
  80, 22
Do : Sleep : Until Me Is Nothing

Sub cmd_Click(Index%)
  Local rs = rtf.SelStart
  Select Index%
  Case 1 : rtf.CharFormat = "bIY360"
  Case 2 : rtf.CharFormat = "iBY720"
  EndSelect
  rtf.SetFocus
  rtf.SelStart = rs
```

```
EndSub
```

Remarks

These properties retrieve and set Sel* properties in one step.

See Also

RichEdit

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

DisableNoScroll Property

Purpose

Returns or sets a value that determines whether scroll bars in a **ListBox**, **ComboBox**, or **RichEdit** control are disabled (for other controls, see Remarks below).

Syntax

object**.DisableNoScroll** [**=** boolean]

Description

DisableNoScroll determines whether or not the scroll bars are enabled.

False = (Default) Scroll bars appear normally when displayed.

True = Scroll bars appear dimmed when displayed.

Example

```
Ocx ListBox lb = "", 10, 10, 100, 100
Local n : For n = 1 To 9 : lb.AddItem "Item " & n
  : Next n
Ocx Command cmd = "Disable Scroll", 20, 120, 80,
  22
Do : Sleep : Until Me Is Nothing

Sub cmd_Click
  lb.DisableNoScroll = (cmd.Caption = "Disable
    Scroll")
  cmd.Caption = (lb.DisableNoScroll ? "Enable
   Scroll" : "Disable Scroll")
  If Not lb.DisableNoScroll Then Local n : For n =
   1 To 9 : lb.AddItem "Item " & n : Next n
EndSub
```

Remarks

In **RichEdit** the **DisableNoScroll** property is ignored when the **ScrollBars** property is set to 0 (None). However, when **ScrollBars** is set to 1 (Horizontal), 2 (Vertical), or 3 (Both), individual scroll bars are disabled when there are too few lines of text to scroll vertically or too few characters of text to scroll horizontally.

To reproduce this function in other objects, such as a Form, you can use the built-in EnableScrollBar() API, which has the added advantage of allowing you to disable (or enable) only part of the scroll bar if that is what you wish. An example of how to use the API is below (the constants listed are also 'built-in' and are included in this example only for illustrative purposes):

```
Const SB_HORZ = 0    ' horizontal scrollbar
Const SB_VERT = 1    ' vertical scrollbar
Const SB_CTL = 2     ' scollbar control
Const SB_BOTH = 3    ' both horiz & vert scrollbars
Const ESB_ENABLE_BOTH = &H0    ' enable both arrows
Const ESB_DISABLE_LTUP = &H1   ' disable left/up
  arrows
Const ESB_DISABLE_RTDN = &H2   ' disable right/down
  arrows
Const ESB_DISABLE_BOTH = &H3   ' disable both
  arrows
OpenW 1 : Win_1.ScrollBars = 2
~EnableScrollBar(Win_1.hWnd, SB_VERT,
  ESB_DISABLE_BOTH)
```

See Also

ListBox, ComboBox, RichEdit

{Created by Sjouke Hamstra; Last updated: 17/11/2014 by James Gaite}

FormatDC, FormatWidth Properties

Purpose

Returns or sets the information that a **RichEdit** control uses to format its output for a particular device.

Syntax

RichEdit.**FormatDC** [= hDC]

RichEdit.**FormatWidth** [= width]

hDC:Handle
width:Long, in twips

Description

The **FormatDC** property sets the target device to format for. It uses the paper width to format a rich edit control's contents for that device, such as a printer. This is useful for WYSIWYG (what you see is what you get) formatting, in which an application positions text using the printer's font metrics instead of the screen's.

FormatWidth allows you to specify the line width for which a rich edit control formats its text.

Example

```
Ocx RichEdit rtf = "", 10, 10, 300, 200
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelItalic = 1
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelBold = 1
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelItalic = 0
rtf.SelText = String( 5, "GFA-BASIC 32 ")
Message "Click here' to effect changes"
rtf.FormatDC = Printer.hDC
rtf.FormatWidth = Printer.Width
Do : Sleep : Until Me Is Nothing
```

See Also

RichEdit, SelPrint

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

SelBold, SelItalic, SelFontName, SelFontSize, SelStrikeout, SelUnderline, SelColor, SelCharOffset Properties

Purpose

Return or set font color and styles for a **RichEdit** control in the following formats: **Bold**, *Italic*, Strikethru, and Underline. **SelCharOffset** determines the superscript or subscript distance from the baseline.

Syntax

object**.SelBold** [= variant]

object**.SelItalic** [= variant]

object**.SelFontName** [= variant]

object**.SelFontSize** [= variant]

object**.SelStrikeout** [= variant]

object**.SelUnderline** [= variant]

object**.SelColor** [= variant]

object**.SelCharOffset** [= variant]

object:RichEdit Ocx Object

Description

Use these font properties to format the selected text in a **RichEdit** control.

The settings for *variant* are:

NullThe selection or character following the insertion point contains characters that have a mix of the appropriate font styles.

*value***True** or name for **SelFontName,** size in points for **SelFontSize**, and RGB-value for **SelColor**. All the characters in the selection, or character following the insertion point, have the appropriate font style.

False (Default) or empty string for **SelFontName** and size in points for **SelFontSize.** None of the characters in the selection or character following the insertion point have the appropriate font style.

To distinguish between the values of **Null** and **False** when reading these properties at run time, use the **IsNull** function with the **If...Then...Else** statement. See example.

SelCharOffset returns or sets a value that determines whether text appears on the baseline (normal), as a superscript above the baseline, or as a subscript below the baseline. The value can be 0 indicating that the characters appear on the baseline, positive indicating above the baseline, and negative indicating below the baseline (in twips).

Example

```
Global Int32 rdpos
```

```
Ocx RichEdit red = "", 10, 10, 200, 200 :
  .MultiLine = True : .BorderStyle = 1
Ocx Command cmd1 = "Change Font", 230, 10, 120, 22
Ocx Command cmd2 = "Change Colour", 230, 40, 120,
  22
Ocx Option opt(0) = "Subscript", 230, 70, 120, 14
Ocx Option opt(1) = "Normal", 230, 85, 120, 14 :
  opt(1).Value = 1
Ocx Option opt(2) = "Superscript", 230, 100, 120,
  14
red.SetFocus
Do : Sleep : Until Me Is Nothing

Sub cmd1_Click
  Ocx CommDlg cd
  cd.Flags = cdfBoth
  cd.ShowFont
  With red
    .SelFontName = cd.FontName
    .SelFontSize = cd.FontSize
    .SelBold = cd.FontBold
    .SelItalic = cd.FontItalic
    .SelStrikeout = cd.FontStrikethru
    .SelUnderline = cd.FontUnderline
  End With
  red.SetFocus
EndSub

Sub cmd2_Click
  Ocx CommDlg cd
  cd.Flags = cdcFullOpen | cdcRgbInit
  cd.Color = red.SelColor
  cd.ShowColor
  red.SelColor = cd.Color
  red.SetFocus
EndSub
```

```
Sub opt_Click(Index%)
  red.SelCharOffset = 90 * (Index% - 1)
  red.SetFocus
EndSub

Sub red_GotFocus
  red.SelStart = rdpos
EndSub

Sub red_LostFocus
  If Not red Is Nothing Then rdpos = red.SelStart
EndSub
```

To find if some or all of the selected text matches a certain criteria, use the following code:

```
If IsNull(RichEdit1.SelBold) = True Then
  ' Code to run when selection is mixed.
ElseIf RichEdit1.SelBold = False Then
  ' Code to run when selection is not bold.
End If
```

Remarks

Null differs from zero, these properties can only be queried with **IsNull**().

See Also

RichEdit, IsNull

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

SelHangingIndent, SelIndent, SelRightIndent Properties

Purpose

Returns or sets the margin settings for the paragraph(s) in a **RichEdit** control that either contain the current selection or are added at the current insertion point.

Syntax

object.**SelHangingIndent** [= *variant*]

object.**SelIndent** [=*variant*]

object.**SelRightIndent** [= *variant*]

object:RichEdit

Description

These properties return and set a **Variant** (Long) that determines the amount of indent. These properties use the Twips scale mode units.

These properties return **Null** if the selection spans multiple paragraphs with different margin settings.

Example

```
OpenW 1 : AutoRedraw = 1
Global Int32 n, rdpos
```

```
Ocx RichEdit red = "", 10, 10, 200, 200 :
  .MultiLine = True : .BorderStyle = 1
Ocx TextBox tb(1) = "", 230, 10, 60, 14 :
  tb(1).ReadOnly = True : tb(1).BorderStyle = 1  :
  Text 297, 11, "Hanging Indent"
Ocx UpDown up(1) : With up(1) : .BuddyControl =
  tb(1) : .Increment = 50 : .Min = -1000 : .Max =
  1000 : .Value = red.SelHangingIndent : End With
Ocx TextBox tb(2) = "", 230, 30, 60, 14 :
  tb(2).ReadOnly = True : tb(2).BorderStyle = 1  :
  Text 297, 31, "Left Indent"
Ocx UpDown up(2) : With up(2) : .BuddyControl =
  tb(2) : .Increment = 50 : .Min = 0 : .Max = 1000
  : .Value = red.SelIndent : End With
Ocx TextBox tb(3) = "", 230, 50, 60, 14 :
  tb(3).ReadOnly = True : tb(3).BorderStyle = 1  :
  Text 297, 51, "Right Indent"
Ocx UpDown up(3) : With up(3) : .BuddyControl =
  tb(3) : .Increment = 50 : .Min = 0 : .Max = 1000
  : .Value = red.SelRightIndent : End With
For n = 1 To 40 : red.Text = red.Text & "GFA BASIC
  is great " : Next n
Do : Sleep : Until Me Is Nothing

Sub red_GotFocus
  red.SelStart = rdpos
EndSub

Sub red_LostFocus
  If Not red Is Nothing Then rdpos = red.SelStart
EndSub

Sub up_Change(Index%)
  Select Index%
  Case 1 : red.SelHangingIndent = up(1).Value
  Case 2 : red.SelIndent = up(2).Value
  Case 3 : red.SelRightIndent = up(3).Value
  EndSelect
  red.SetFocus
EndSub
```

To find if the current selection contains some (but not all) indented text, use the following code:

```
If IsNull(RichEdit1.SelIndent) = True Then
  ' Code to run when selection is mixed.
Else      ' RichEdit1.SelIndent > 0
  ' Code to run when selection is not mixed.
End If
```

Known Issues

[Fixed OCX v2.36 Build 1905] When you update either **SelHangingIndent** or **SelIndent**, the other assumes the same value. Additionally, the value in either only seems to create a standard left indent - it is not possible to get a hanging indent. There is currently no workaround for this problem.

See Also

RichEdit

{Created by Sjouke Hamstra; Last updated: 20/05/2019 by James Gaite}

TextRTF, SelRTF Properties

Purpose

Returns or sets (selection of) the text of a **RichEdit** control, including all .rtf code.

Syntax

*RichEdit***.TextRTF** [**=** *string*]

*RichEdit***.SelRTF** [**=** *string*]

Description

TextRTF returns or sets the text (in .rtf format).

SelRTF returns or sets the text (in .rtf format) in the current selection. **SelText** returns or sets plain text. Setting new selected text sets **SelLength** to 0 and replaces the selected text with the new string.

Example

```
Global Int32 rdpos
Ocx RichEdit red = "", 10, 10, 200, 200 :
  .MultiLine = True : .BorderStyle = 1
Ocx Command cmd1 = "Change Font", 230, 10, 140, 22
Ocx Command cmd2 = "Change Colour", 230, 40, 140,
  22
Ocx Option opt(0) = "Subscript", 230, 70, 120, 14
Ocx Option opt(1) = "Normal", 230, 85, 120, 14 :
  opt(1).Value = 1
```

```
Ocx Option opt(2) = "Superscript", 230, 100, 120,
  14
Ocx Command cmd3 = "Save Selection to File", 230,
  130, 140, 22
Ocx Command cmd4 = "Save Whole Text to File", 230,
  160, 140, 22
red.SetFocus
Do : Sleep : Until Me Is Nothing
If Exist(App.Path & "\AllText.rtf") Then Kill
  App.Path & "\AllText.rtf"
If Exist(App.Path & "\Select.rtf") Then Kill
  App.Path & "\Select.rtf"

Sub cmd1_Click
  Ocx CommDlg cd
  cd.Flags = cdfBoth
  cd.ShowFont
  With red
    .SelFontName = cd.FontName
    .SelFontSize = cd.FontSize
    .SelBold = cd.FontBold
    .SelItalic = cd.FontItalic
    .SelStrikeout = cd.FontStrikethru
    .SelUnderline = cd.FontUnderline
  End With
  red.SetFocus
EndSub

Sub cmd2_Click
  Ocx CommDlg cd
  cd.Flags = cdcFullOpen | cdcRgbInit
  cd.Color = red.SelColor
  cd.ShowColor
  red.SelColor = cd.Color
  red.SetFocus
EndSub
```

```
Sub cmd3_Click
  If red.SelLength = 0 Then Message "No text has
    been selected"  : Exit Sub
  Local t$ = red.SelRTF : BSave App.Path &
    "\Select.rtf", V:t$, Len(t$)
  Message "Selected text saved as Select.rtf"
EndSub

Sub cmd4_Click
  Local t$ = red.TextRTF : BSave App.Path &
    "\AllText.rtf", V:t$, Len(t$)
  Message "Selected text saved as AllText.rtf"
EndSub

Sub opt_Click(Index%)
  red.SelCharOffset = 90 * (Index% - 1)
  red.SetFocus
EndSub

Sub red_GotFocus
  red.SelStart = rdpos
EndSub

Sub red_LostFocus
  If Not red Is Nothing Then rdpos = red.SelStart
EndSub
```

Remarks

You can use the **TextRTF** and **SelRTF** properties along with **Open**/**Print #** to write .rtf files.

See Also

RichEdit

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

SelTabCount, SelTabs Properties

Purpose

Returns or sets the number of tabs and the absolute tab positions of text in a **RichEdit** control.

Syntax

object.**SelTabCount** [= *variant*]

object.**SelTabs**(index%) [=*variant*]

object:RichEdit

Description

The **SelTabCount** property determines the number of tab positions in the selected paragraph(s) or in those paragraph(s) following the insertion point.

SelTabs(index%) identifies a specific tab. The first tab location has an index of zero (0). The last tab location has an index equal to **SelTabCount** minus 1.

These properties return **Null** if the selection spans multiple paragraphs with different tab settings.

Example

```
Local i%
Ocx RichEdit red = "", 10, 10, 300, 300 :
  .MultiLine = True : .BorderStyle = 1 :
```

```
  .WantSpecial = True
red.SelTabCount = 5
For i% = 0 To .SelTabCount - 1
  red.SelTabs(i%) = 3 * i%
Next
Do : Sleep  : Until Me Is Nothing
```

Remarks

Null differs from zero, these properties can only be queried with **IsNull**().

By default, pressing TAB when typing in a **RichEdit** control causes focus to move to the next control in the tab order, as specified by the **TabIndex** property. One way to insert a tab in the text is by pressing CTRL+TAB. However, users who are accustomed to working with word processors may find the CTRL+TAB key combination contrary to their experience. You can enable use of the TAB key to insert a tab in a **RichEdit** control by setting **WantSpecial** = 1.

See Also

RichEdit, IsNull

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Find Method

Purpose

Searches the text in a **RichEdit** control for a given string.

Syntax

RichEdit.**Find**(search$,start [,end = -1] [, options])

Description

The **Find** method searches for a string. The *start* parameter (optional) is an integer character index that determines where to begin the search. Each character in the control has an integer index that uniquely identifies it. The first character of text in the control has an index of 0. The optional *end* parameter determines where to end the search. The *options* parameter specifies how to perform the search:

rtfUp(1)	Searches upwards. Only supported with RichEd20.Dll, not with RichEd32.Dll which was part of the first Windows 95.
rtfWholeWord(2)	Find whole word.
rtfMatchCase(4)	Only exact match
rtfFindNext(8)	Searches the next match.

If the text searched for is found, the **Find** method highlights the specified text and returns the index of the first character highlighted. If the specified text is not found, the **Find** method returns -1.

Example

```
OpenW 1
Me.Sizeable = 0
Ocx Timer tmr : .Interval = 500 : .Enabled = 1
Ocx RichEdit rtf = , 0, _Y / 2, _X, _Y / 2
rtf.HideSelection = 0
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelItalic = 1
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelBold = 1
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelItalic = 1
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelStart = 1 : rtf.SelLength = 0
Do
  Sleep
Loop Until Me Is Nothing

Sub tmr_Timer
  Local Int l
  Static Int direction = 0
  l = rtf.Find("BASIC", rtf.SelStart + 1, -1,
    direction)
  If l < 0 Then  direction = (direction == 0 ? 1 :
    0)
End Sub
```

Remarks

See Also

RichEdit

{Created by Sjouke Hamstra; Last updated: 06/10/2014 by James Gaite}

LoadFile, SaveFile Methods (RichEdit)

Purpose

Loads and saves an .rtf file or text file into a **RichEdit** control.

Syntax

RichEdit.**LoadFile** *filename* [, *filetype*]

RichEdit.**SaveFile**(*filename*[, *filetype*])

filename:sexp
filetype:iexp

Description

LoadFile loads an .rtf file (default) or text file specified in *filename* and replaces the entire contents of the rich edit control. The optional *filetype* can be **rtfRTF** (0) or **rtfText** (1).

SaveFile saves the contents (as an rtf file by default) of a **RichEdit** control to a file *filename*. The optional *filetype* can be **rtfRTF** (0) or **rtfText** (1) to save it as a text file.

Example

```
Ocx RichEdit rtf = "", 10, 10, 300, 200 :
  .BorderStyle = 3
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelItalic = 1
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelBold = 1
rtf.SelText = String( 5, "GFA-BASIC 32 ")
rtf.SelItalic = 0
rtf.SelText = String( 5, "GFA-BASIC 32 ")
Ocx RichEdit rtf_copy = "", 320, 10, 300, 200 :
  .BorderStyle = 3
Ocx Command cmd = "Save RichEdit object", 50, 220,
  200, 22
Do : Sleep : Until Me Is Nothing

Sub cmd_Click
  Static opt|, f$
  Inc opt|
  Switch opt|
  Case 1
    Ocx CommDlg cd
    cd.ShowSave
    f$ = cd.FileName
    rtf.SaveFile f$, rtfRTF
    cmd.Caption = "Load into second RichEdit
      object"
  Case 2
    rtf_copy.LoadFile f$
    cmd.Caption = "Close window"
  Case 3
    Kill f$
    Me.Close
  EndSwitch
EndSub
```

Remarks

You can also use **Input** and the **TextRTF** and **SelRTF** properties to read .rtf files. For example:

```
Open "mytext.rtf" For Input As #1

rft1.TextRTF = Input$(LOF(1), 1)

Close #1
```

See Also

RichEdit, TextRTF, SelRTF

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Span, UpTo Methods

Purpose

Span selects text in a **RichEdit** control based on a set of specified characters. **UpTo** moves the insertion point up to, but not including, the first character that is a member of the specified character set.

Syntax

RichEdit.**Span**(*characterset*[,*forward = 1*] [,*negate = 0*])

RichEdit.**UpTo**(*characterset* [,*forward = 1*] [,*negate = 0*])

characterset:sexp
forward, negate:Boolean exp

Description

The *characterset* specifies the set of characters to look for when extending the selection (Span) or moving the selection point (UpTo), based on the optional value of *negate*.

By default *negate* = False and **Span** selects the characters which appear in the *characterset* argument. The selection stops at the first character found that does not appear in the *characterset* argument.
Setting *negate* = True will select the characters which do not appear in the *characterset* argument. The selection stops at the first character found that appears in the *characterset* argument.

For **UpTo** (*negate* = 0) the characters specified in the *characterset* argument are used to move the insertion point. Setting *negate* = True will use the characters specified in the *characterset* argument to move the insertion point.

Example

```
Sub rtf1_KeyUp(Code&, Shift&)
  Select Code
  Case Asc("S")         ' Alt+S or Ctrl+S
    ' Move insertion point to the end of the
      sentence.
    If Shift = 4 rtf1.UpTo ".?!:"
    ' Select to the end of the sentence.
    If Shift = 2 rtf1.Span ".?!:", True, True
  Case Asc("W")         ' Alt+W or Ctrl+S
    ' Move insertion point to the end of the word.
    If Shift = 4 rtf1.UpTo " ,;:.?!"
    ' Select to the end of the word.
    If Shift = 2 rtf1.Span " ,;:.?!", True, True
  End Select
End If
```

See Also

RichEdit

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

ColorFormat, ImageHeight, ImageWidth Properties

Purpose

These properties define the **ImageList** creation parameters.

Syntax

ImageList.**ColorFormat** [= flags]

ImageList.**ImageHeight** [= h]

ImageList.**ImageWidth** [= w]

flags, h, w:iexp

Description

The properties describe the type of image list to create. After adding the first image to the control, these properties are read-only. The **ImageList** control is not created before the first item is added.

ColorFormat = *flags* specifies a value how to create the image list. This can be one of the following flags:

0	Device dependent bitmap with a mask.
4	Use 4-bpp (bits-per-pixel) DIB Section.
8	Use 8-bpp DIB Section.
16	Use 16-bpp DIB Section.
24	Use 24-bpp DIB Section.

32   Use 32-bpp DIB Section.

By adding 1 to the *flags* value the respective image list is created without a mask; *flags* = 1, 5, 9, 17, 25, 33 do create a mask.

When the first image added contains a palette, the **ImageList** control creates a list with a palette.

The **ImageHeight** and **ImageWidth** properties define the height and width of the images in pixels.

Example

```
OpenW 1, 30, 30, 300, 300
Cls colBtnFace
Ocx ImageList iml
iml.ImageWidth = 32
iml.ImageHeight = 32
iml.ColorFormat = 1
iml.MaskColor = $0c0c0c
iml.UseMaskColor = True
//iml.ListImages.Add , "new", LoadPicture(":new")
 - Requires an inline picture titled ":new"
iml.ListImages.Add , "app",
 CreatePicture(LoadIcon(Null, IDI_APPLICATION))
Dim p As Picture : Set p =
 iml.ListImage(1).ExtractIcon
PaintPicture p, 0, 0
Do : Sleep : Until Win_1 Is Nothing
```

Remarks

When the **ImageList** control is bound to another Ocx control, all images in the **ListImages** collection - no matter what their size - will be displayed in the second (bound control) at the size specified by the **ImageHeight** and **ImageWidth** properties.

See Also

ImageList, ListImages

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

MaskColor, UseMaskColor Properties (ImageList)

Purpose

Determines how and if a masked bitmap is created for the next image of an **ImageList** control.

Syntax

ImageList.**MaskColor** [= rgb]

ImageList.**UseMaskColor** [= Boolean]

rgb:iexp

Description

When **UseMaskColor** is set, new images have their mask added to the list as well. Of course, the ImageList control must be created using the correct **ColorFormat** value to have masked images in the first place.

With **MaskColor** you specify a color to be combined with the image bitmap to generate the masks. When you do this, each pixel of the specified color in the image bitmap is changed to black, and the corresponding bit in the mask is set to one. This results in transparency for any pixel in the image that matches the specified color when the image is drawn. That is to say, transparency is only obtained in a ListImage.**Draw** or ImageList.**Overlay** operation.

Example

```
OpenW 1, 30, 30, 300, 300 : Win_1.AutoRedraw = 1
Cls colBtnFace
Ocx ImageList iml : .ImageWidth = 32 :
  .ImageHeight = 32
iml.ColorFormat = 1
iml.MaskColor = $0c0c0c
iml.UseMaskColor = True
iml.ListImages.Add , "new",
  CreatePicture(LoadIcon(Null, IDI_WARNING), False)
iml.ListImages(1).Draw Me.hDC, 0, 30, 0
iml.ListImages.Item(1).Draw Me.hDC, 50, 30, 1
iml.ListImage(1).Draw Me.hDC, 100, 30, 2
Dim lim As ListImage
Set lim = iml.ListImage(1)
lim.Draw Me.hDC, 150, 30, 3
```

Remarks

The **UseMaskColor** and **MaskColor** properties can only be modified before a single image is added to the image list, it is not a general setting.

The **UseMaskColor** and **MaskColor** properties have no proper function when the **ImageList** is used as an image repository for **ToolBar** buttons. The **ToolBar** control simply draws the non-masked images on the button surface.

See Also

ImageList, ListImage

{Created by Sjouke Hamstra; Last updated: 13/10/2014 by James Gaite}

Add, AddItem, AddPart Method (ImageList, ListImages)

Purpose

Adds a **ListImage** to a **ListImages** collection in an **ImageList** control and returns a reference to the newly created **ListImage** object.

Syntax

*ImageList***.Add[Item]**([index], [key], picture)

*ListImages***.Add**([index], [key], picture)

*ImageList.***AddPart** [index], [key], [picture], [X], [Y]

index, key, picture: Variant exp
x, y:iexp

Description

Add, **AddItem**, *ListImages*.**Add**, and **AddPart** perform the same task; they add a single image to the **ListImages** collection owned by the **ImageList** control.

You can load either bitmaps, cursors, or icons into a **ListImage** object. To load a bitmap or icon, you can use the **LoadPicture** function.

index	Optional. An integer specifying the position where you want to insert the **ListImage.** If no

	index is specified, the **ListImage** is added to the end of the **ListImages** collection.
key	Optional. A unique string expression that can be used to access a member of the collection.
picture	Specifies the picture to be added to the collection.

AddPart adds a part of an image to the collection. The x, y coordinate specify the location of the piece to be grabbed from a picture. The width and the height are determined by the dimensions of the ImageList control.

Example

```
Const IDI_SHIELD = 32518
Global Int32 m, n, x, y
Dim h As Handle, p As Picture
Ocx ImageList iml
iml.ImageWidth = 32
iml.ImageHeight = 32
Dim lim As ListImage
iml.ListImages.Add , "app",
  CreatePicture(LoadIcon(Null, IDI_APPLICATION))
iml.ListImages.Add , "info",
  CreatePicture(LoadIcon(Null, IDI_INFORMATION))
Set lim = iml.ListImages.Add( , "error",
  CreatePicture(LoadIcon(Null, IDI_ERROR)))
iml.Add , "query", CreatePicture(LoadIcon(Null,
  IDI_QUESTION))
iml.AddItem , "security",
  CreatePicture(LoadIcon(Null, IDI_SHIELD))
iml.AddPart , "winlogo",
  CreatePicture(LoadIcon(Null, IDI_WINLOGO))
For m = 0 To 5 : x = (m * 32) : y = (m * 32)
  For n = 1 To 6
```

```
    Set p = iml(n).ExtractIcon : PaintPicture p, x,
      y
    Add y, 32 : If y > 191 Then y = 0
  Next n
Next m
'
// AddPart example
'
// Loads the 36 icons in reverse order into a new
  ImageList...
// ...and displays them to the right of the
  originals
Get 0, 0, 191, 191, h
Set p = CreatePicture(h, False)
Ocx ImageList iml1
iml1.ImageHeight = 32
iml1.ImageWidth = 32
// or could be calculated like this:
// iml1.ImageHeight = HimetsToPixelX(p.Height) / 6
// iml1.ImageWidth = HimetsToPixelX(p.Width) / 6
For n = 5 To 0 Step -1
  For m = 5 To 0 Step -1
    iml1.AddPart , , p, (n * 32), (m * 32)
  Next m
Next n
For m = 0 To 5 : x = (m * 32) + 300 : y = 0
  For n = 1 To 6
    Set p = iml1((m * 6) + n).ExtractIcon :
      PaintPicture p, x, y
    Add y, 32 : If y > 200 Then y = 0
  Next n
Next m
Set p = Nothing
Do : Sleep : Until Me Is Nothing
```

Remarks

GFA-BASIC 32 specific

Instead of explicitly using the **ListImages** collection to access a **ListImage** element, you can use a shorter notation. First, the **ImageList** supports an **Item** property:

iml.**Item**(idx)iml.**ListImages**.**Item**(idx)

Like the **Item** method of iml.**ListImages, Item** is the default method of **ImageList**. Therefore, a **ListImage** can be accessed as follows:

iml(idx)iml.**ListImages**(idx)

iml!idximl.**ListImages**!idx

Each dot saves about 30 bytes of code.

To enumerate over the **ListImages** collection of an **ImageList** Ocx, use For Each on the Ocx control directly, like:

```
Local lim As ListImage, iml As ImageList
For Each lim In iml : DoSomething(lim) : Next
```

See Also

ImageList, ListImages, ListImage

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

OverLay Method (ImageList)

Purpose

Draws one image from a **ListImages** collection over another, and returns the result.

Syntax

ImageList.**Overlay (***index1*, *index2***)**

index1, index2:Variant

Description

The **Overlay** method combines two images and returns a new **Picture** object. *index1* is an integer (**Index** property) or unique string (**Key** property) that specifies the image to be overlaid. *index2* specifies the image to be drawn over the object specified in *index1*. The color of the image that matches the **MaskColor** property is made transparent. If no color matches, the image is drawn opaquely over the other image.

Example

```
// Create two pictures to be overlaid
OpenW 1 : Win_1.AutoRedraw = 1
Color 255 : Draw 0, 0 To 100, 0 To 0, 100 To 0, 0
  : Fill 10, 10
Color RGB(255, 255, 255), 255 : Text 10, 10,
  "GFABasic"
Color RGB(0, 255, 0) : Draw 201, 0 To 201, 100 To
  101, 100 To 201, 0 : Fill 190, 90
```

```
Color RGB(255, 255, 0), RGB(255, 255, 255) : Text
  111, 10, "GFABasic"
Local Handle h1, h2 : Local Picture p1, p2, p3, p4
Get 0, 0, 100, 100, h1 : Set p1 =
  CreatePicture(h1, False)
Get 101, 0, 200, 100, h2 : Set p2 =
  CreatePicture(h2, False)
Cls
// Create Imagelist
Ocx ImageList iml
iml.MaskColor = RGB(255, 255, 255)
iml.UseMaskColor = True
iml.Add , "First", p1
iml.Add , "Second", p2
// Draw the two listimages
// (Note the 'greyed' transparent area)
iml(1).Draw Win_1.hDC, 0, 0
iml(2).Draw Win_1.hDC, 105, 0
// Create the composite image, first by using the
  index...
Set p3 = iml.Overlay(1, 2)
// ...and then by using the unique Key.
Set p4 = iml.Overlay("First", "Second")
// Display the two composite images
PaintPicture p3, 0, 150
PaintPicture p4, 105, 150
```

Remarks

Use the **Overlay** method in conjunction with the **MaskColor** property to create a single image from two disparate images. The **Overlay** method imposes one bitmap over another to create a third, composite image. The **MaskColor** property determines which color of the overlaying image is transparent.

See Also

ImageList, ListImages, Picture

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Style, LineStyle, Indentation Property (TreeView)

Purpose

Style returns or sets the type of graphics (images, text, plus/minus, and lines) and text that appear for each **Node** object in a **TreeView** control.

LineStyle returns or sets the style of lines displayed between **Node** objects.

Indentation returns or sets the width of the indentation of the **Node** objects.

Syntax

TreeView.**Style** [= *value%*]

TreeView.**LineStyle** [= *value%*]

TreeView.**Indentation** [= *single*]

Description

The **Style** property can take one of the tvw-constant values.

tvwTextOnly (0) - Text only.

tvwPictureText (1) - Image and text.

tvwPlusMinusText (2) - Plus/minus and text.

tvwPlusPictureText (3) - Plus/minus, image, and text.

tvwTreeLinesText (4) - Lines and text.

tvwTreeLinesPictureText (5) - Lines, image, and text.

tvwTreeLinesPlusMinusText (6) - Lines, plus/minus, and text.

tvwTreeLinesPlusMinusPictureText (7) - (Default) Lines, plus/minus, image, and text.

If the **Style** property is set to a value that includes lines, the **LineStyle** property determines the appearance of the lines. If the **Style** property is set to a value that does not include lines, the **LineStyle** property will be ignored.

The **LineStyle** property can take one of the tvw-constant values.

tvwTreeLines (0) - (Default) Tree lines. Displays lines between Node siblings and their parent Node.

tvwRootLines (1) - Root Lines. In addition to displaying lines between Node siblings and their parent Node, also displays lines between the root nodes.

You must set the **Style** property to a style that includes tree lines.

The **Indentation** property specifies the width (in pixels) that each object is indented. If you change the **Indentation** property at run time, the **TreeView** is redrawn to reflect the new width. The property value cannot be negative.

Example

```
Global a$, n As Int32
Ocx ImageList iml : iml.ImageHeight = 16 :
  iml.ImageWidth = 16
For n = 1 To 7 : iml.Add , ,
  CreatePicture(LoadIcon(Null, 32511 + n)) : Next n
Ocx TreeView tv = "", 10, 10, 200, 300 :
  tv.ImageList = iml
For n = 1 To 7
  If Odd(n)
    tv.Add , , , "Icon" & n, n, n
  Else
    tv.Add n - 1, tvwChild, , "Icon" & n, n :
      tv(n).EnsureVisible
  EndIf
Next n
Text 220, 13, "Style:" : Ocx ComboBox cb1 = "",
  280, 10, 200, 22  : cb1.Sorted = False
For n = 0 To 7 : Read a$ : cb1.AddItem a$, n :
  Next n
cb1.ListIndex = tv.Style
Text 220, 43, "Line Style:" : Ocx ComboBox cb2 =
  "", 280, 40, 200, 22 : cb2.Sorted = False
For n = 0 To 1 : Read a$ : cb2.AddItem a$, n :
  Next n
cb2.ListIndex = tv.LineStyle
Text 220, 73, "Indentation:" : Ocx TextBox tb =
  "", 280, 72, 40, 14 : tb.BorderStyle = 1 :
  tb.ReadOnly = True
Ocx UpDown up : up.BuddyControl = tb  : .Max = 40
  : .Value = tv.Indentation
Do : Sleep : Until Me Is Nothing

Sub cb1_Click
  If tv.Style <> cb1.ItemData(cb1.ListIndex) Then
    tv_Redraw
EndSub
```

```
Sub cb2_Click
  If tv.LineStyle = cb2.ItemData(cb2.ListIndex)
    Then tv_Redraw
EndSub

Sub up_Change
  tv.Indentation = up.Value
EndSub

Sub tv_Redraw
  // Changing the Style and LineStyle values once
    the treeview has...
  // ...been drawn can lead to some odd results, so
    it is necesaary...
  // ...to redraw it upon every change to show it
    correctly.
  Set tv = Nothing
  Ocx TreeView tv = "", 10, 10, 200, 300 :
    tv.ImageList = iml
  tv.Style = cb1.ItemData(cb1.ListIndex)
  tv.LineStyle = cb2.ItemData(cb2.ListIndex)
  For n = 1 To 7
    If Odd(n)
      tv.Add , , , "Icon" & n, n, n
    Else
      tv.Add n - 1, tvwChild, , "Icon" & n, n :
        tv(n).EnsureVisible
    EndIf
  Next n
EndSub
Data
  tvwTextOnly,tvwPictureText,tvwPlusMinusText,tvwPl
  usPictureText,tvwTreeLinesText
Data
  tvwTreeLinesPictureText,tvwTreeLinesPlusMinusText
  ,tvwTreeLinesPlusMinusPictureText
Data tvwTreeLines,tvwRootLines
```

See Also

TreeView

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

LabelEdit Property, StartLabelEdit Method

Purpose

Returns or sets a value that determines if a user can edit labels of **ListItem** or **Node** objects in a **ListView** or **TreeView** control.

Syntax

object.**LabelEdit** [= *integer*]

object.**StartLabelEdit**

object:ListView, TreeView

Description

Label editing of an object is initiated when a selected object is clicked (if the **LabelEdit** property is set to Automatic). That is, the first click on an object will select it; a second (single) click on the object will initiate the label editing operation.

LabelEdit can have the following values:

0 - Automatic (Default). The BeforeLabelEdit event is generated when the user clicks the label of a selected node.

1 - Manual. The BeforeLabelEdit event is generated only when the **StartLabelEdit** method is invoked.

The **LabelEdit** property, in combination with the **StartLabelEdit** method, allows you to programmatically determine when and which labels can be edited. When the **LabelEdit** property is set to 1, no label can be edited unless the **StartLabelEdit** method is invoked.

Example

```
Global li As ListItem, n As Int32
OpenW 1
Ocx ListView lv = "", 10, 10, 200, 300 : .View = 3
  : .GridLines = True : .FullRowSelect = True
lv.ColumnHeaders.Add , , "Column1" :
  lv.ColumnHeaders.Add , , "Column2"
For n = 1 To 20
  lv.ListItems.Add , n , "Item " & Format(n, "00")
Next n
Ocx Command cmd1 = "Disable Label Editing", 220,
  10, 140, 22
Ocx Command cmd2 = "Manually Edit Selected Item",
  220, 35, 140, 22
Do : Sleep : Until IsNothing(Win_1)

Sub cmd1_Click
  cmd1.Caption = (lv.LabelEdit = 1 ? "Disable" :
    "Enable") & " Label Editing"
  lv.LabelEdit = 1 - lv.LabelEdit
EndSub

Sub cmd2_Click
  If lv.SelectedCount <> 0
    lv.SetFocus
    lv.StartLabelEdit  // Error: Only works when
      LabelEdit = 0
  EndIf
EndSub
```

Known Issues

Rather than opening a label for editing for all values of **LabelEdit** as happens in VB6, **StartLabelEdit** only seems to work when **LabelEdit** = 0.

See Also

ListView, TreeView, BeforeLabelEdit, AfterLabelEdit

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

SelectedItem Property, SelectedIndex Property (TabStrip)

Purpose

Returns a reference to a selected **ListItem**, **Node**, or **Tab** object.

Syntax

object.**SelectedItem** [= *item*]

TabStrip.**SelectedIndex** [= *integer*]

object:TreeView, ListView, TabStrip
item:ListItem, Node, Tab

Description

SelectedItem returns a reference to a selected **ListItem, Node**, or **Tab** object, or sets a specified **ListItem, Node**, or **Tab** to a selected state.

This property is typically used to return a reference to a **ListItem, Node**, or **Tab** or object that the user has clicked or selected. With this reference, you can validate an object before allowing any further action, as demonstrated in the example.

To programmatically select a **ListItem** object, you can (optionally) use the **Set** statement with the **SelectedItem** property, as follows:

Set ListView.SelectedItem = ListView.ListItems(1)

ListView.SelectedItem = ListView.ListItems(1)

GFA-BASIC 32 allows both versions.

The **TabStrip** control property **SelectedIndex** returns or sets the **Tab** object by index (number). See TabStrip_Change event for an example.

Example

```
Global Int32 n
OpenW 1 : AutoRedraw = 1
Ocx TabStrip tbs = "", 0, 10,
  TwipsToPixelX(Win_1.Width) - 20, 40
Text 10, 70, "Select: "
Ocx ComboBox cmb = "", 50, 67, 100, 22 : .Sorted =
  False
For n = 1 To 20 : tbs.Add n, "Key " & Chr(64 + n),
  "Tab" & n : cmb.AddItem "Tab" & n, n : Next n
cmb.ListIndex = 0
Do : Sleep : Until Win_1 Is Nothing

Sub cmb_Click
  tbs.Tab(cmb.ItemData(cmb.ListIndex)).Selected =
    True
EndSub

Sub tbs_Change
  cmb.ListIndex = tbs.SelectedIndex - 1
  Text 10, 90, "Current Key: " &
    tbs.SelectedItem.Key & "    "
EndSub
```

See Also

TreeView, ListView, TabStrip, ListItem, Node, Tab

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Sorted Property (TreeView, Node)

Purpose

Returns or sets a value that determines whether the child nodes of a **Node** object are sorted alphabetically.

Returns or sets a value that determines whether the root level nodes of a **TreeView** control are sorted alphabetically.

Syntax

TreeView.**Sorted** [= *boolean*]

Node.**Sorted** [= *boolean*]

Description

When set to True, the **Node** objects are sorted alphabetically by their **Text** property. **Node** objects whose **Text** property begins with a number are sorted as strings, with the first digit determining the initial position in the sort, and subsequent digits determining sub-sorting. If False, the **Node** objects are not sorted.

The **Sorted** property can be used in two ways: first, to sort the **Node** objects at the root (top) level of a **TreeView** control and, second, to sort the immediate children of any individual **Node** object.

Setting the **Sorted** property to **True** sorts the current **Nodes** collection only. When you add new **Node** objects to a **TreeView** control, you must set the **Sorted** property to **True** again to sort the added **Node** objects.

Example

```
Ocx TreeView tv1 = "", 250, 10, 230, 200

Dim node As Node

Set node = tv1.Nodes.Add(,,,"Parent Node")

node.Sorted = True

tv1.Sorted = True ' Top level Node objects are sorted.
```

See Also

TreeView, Nodes, Node

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Add, AddItem Method (TreeView, Nodes)

Purpose

Adds a **Node** to a **Nodes** collection in a **TreeView** control and returns a reference to the newly created **Node** object.

Syntax

*TreeView***.Add[Item]**(relative, relationship, key, text, image, selectedimage)

*Nodes***.Add**(relative, relationship, key, text, image, selectedimage)

relative, relationship, key, text, image, selectedimage: Variant exp

Description

The **TreeView** Ocx has the **AddItem** and **Add** methods, which act exactly the same. The **Nodes** object supports the **Add** method only.

relative	Optional. The index number or key of a pre-existing Node object. The relationship between the new node and this pre-existing node is found in the next argument, relationship.
relationship	Optional. Specifies the relative placement of the Node object: **tvwFirst** (0) First. The Node is placed

	before all other nodes at the same level of the node named in relative.
	tvwLast (1) Last. The Node is placed after all other nodes at the same level of the node named in relative. Any Node added subsequently may be placed after one added as Last.
	tvwNext (2) (Default) Next. The Node is placed after the node named in relative.
	tvwPrevious (3) Previous. The Node is placed before the node named in relative.
	tvwChild (4) Child. The Node becomes a child node of the node named in relative.
key	Optional. A unique string that can be used to retrieve the Node with the Item method.
text	Required. The string that appears in the Node.
image	Optional. The index of an image in an associated ImageList control.
selectedimage	Optional. The index of an image in an associated ImageList control that is shown when the Node is selected.

Use the **Key** property to reference a member of the **Nodes** collection if you expect the value of an object's **Index** property to change, such as by dynamically adding objects to or removing objects from the collection. The **Nodes** collection is a 1-based collection.

As a **Node** object is added it is assigned an index number, which is stored in the **Node** object's **Index** property. This value of the newest member is the value of the **Node** collection's **Count** property.

Because the **Add** method returns a reference to the newly created **Node** object, it is most convenient to set properties of the new **Node** using this reference. The following example adds several **Node** objects with identical properties:

Example

```
Dim node As Node
Ocx TreeView tv = "", 10, 10, 100, 200
Set node = tv.Add( , tvwChild, "David" , "David")
Set node = tv.Add(1, tvwChild, "Peter", "Peter")
Set node = tv.Add("David", tvwChild, "Angela",
  "Angela")
Set node = tv.Add( , , "Arthur", "Arthur")
tv.Item("Peter").EnsureVisible ' Expand tree to
  see all nodes.
Do : Sleep : Until Me Is Nothing
```

Remarks

GFA-BASIC 32 specific

Instead of explicitly using the **Nodes** collection to access a **Node** element, you can use a shorter notation. First, the **TreeView** supports an **Item** property:

tv.**Item**(idx)tv.**Nodes**.**Item**(idx)

Like the **Item** method of tv.**Nodes, Item** is the default method of **TreeView**. Therefore, a **Node** can be accessed as follows:

tv(idx)tv.**Nodes**(idx)

tv!idxtv.**Nodes**!idx

Each dot saves about 30 bytes of code.

To enumerate over the **Nodes** collection of a **TreeView** Ocx, use For Each on the Ocx control directly, like:

```
Local node1 As Node
For Each node1 In tv : DoSomething(node1) : Next
```

See Also

TreeView, Node, Nodes

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

BeforeLabelEdit, AfterLabelEdit Events

Purpose

BeforeLabelEdit occurs when a user attempts to edit the label of the currently selected **ListItem** or **Node** object. **AfterLabelEdit** occurs after a user edits the label of the currently selected **Node** or **ListItem** object.

Syntax

Sub *object*_**AfterLabelEdit**(Cancel?, NewString As Variant)

Sub *object*_**BeforeLabelEdit**(Cancel?)

object:ListView, TreeView

Description

Both the AfterLabelEdit and the BeforeLabelEdit events are generated only if the **LabelEdit** property is set to 0 (Automatic), or if the **StartLabelEdit** method is invoked.

The BeforeLabelEdit event occurs after the standard **Click** event.

To begin editing a label, the user must first click the object to select it, and click it a second time to begin the operation. The BeforeLabelEdit event occurs after the second click. To determine which object's label is being edited, use the **SelectedItem** property.

The AfterLabelEdit event is generated after the user finishes the editing operation, which occurs when the user clicks on another **Node** or **ListItem** or presses the ENTER key.

To cancel a label editing operation, set *cancel* to any nonzero number or to **True**. If a label editing operation is canceled, the previously existing label is restored.

The *newstring* argument can be used to test for a condition before canceling an operation. The *newstring* Variant is **Null** if the user canceled the operation.

Example

```
Ocx TreeView tv = "", 10, 10, 100, 200
tv.Add , , , "Fred"
tv.Add , , , "Harry"
tv.Add , , , "Archie"
Do : Sleep : Until Me Is Nothing

Sub tv_BeforeLabelEdit(Cancel?)
  If tv.SelectedItem.Index = 1 Then
    MsgBox("This node cannot be edited")
    Cancel = True    ' Cancel the operation
  End If
EndSub

Sub tv_AfterLabelEdit(Cancel?, NewString As
  Variant)
  If IsNumeric(NewString) Then
    MsgBox "No numbers allowed"
    Cancel = True
  End If
EndSub
```

The code checks the index of a selected **Node** before allowing an edit. If the index is 1, the operation is cancelled.

Then the edit is cancelled when if *newstring* is a number.

See Also

ListView, TreeView, LabelEdit, StartLabelEdit

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Expand, Collapse, NodeClick Event (TreeView)

Purpose

The **Expand** event occurs when a **Node** object in a **TreeView** control is expanded, that is, when its child nodes become visible.

The **Collapse** event is generated when any **Node** object in a **TreeView** control is collapsed.

Syntax

Sub *TreeView*_**Expand**(*Node* As Node)

Sub *TreeView*_**Collapse**(*Node* As Node)

Sub *TreeView*_**NodeClick**(*Node* As Node)

Description

The **Expand** event occurs after the Click and DblClick events.

The **Expand** event is generated in three ways: when the user double-clicks a **Node** object that has child nodes; when the **Expanded** property for a **Node** object is set to **True**; and when the plus/minus image is clicked.

The **Collapse** event occurs before the standard Click event.

There are three methods of collapsing a **Node**: by setting the **Node** object's **Expanded** property to **False**, by double-clicking a **Node** object, and by clicking a plus/minus image when the **TreeView** control's **Style** property is set to a style that includes plus/minus images. All of these methods generate the **Collapse** event.

The **NodeClick** event occurs when a **Node** object is first clicked; if you continue to click it, this event does not re-occur, until you click another node and then return to it. The **NodeClick** event occurs before the standard Click event.

The standard Click event is generated when the user clicks any part of the **TreeView** control outside a node object. The **NodeClick** event is generated when the user clicks a particular **Node** object; the **NodeClick** event also returns a reference to a particular **Node** object which can be used to validate the **Node** before further action is taken.

Example

```
Ocx TreeView tv = "", 10, 10, 200, 400
tv.LineStyle = tvwRootLines
tv.Style = tvwPlusMinusText
tv.Add , , , "David"
tv.Add 1, tvwChild, , "Mary"
tv.Add 1, tvwChild, , "Harold"
tv.Add 1, tvwNext, , "Mildred"
tv.Add 4, tvwChild, , "Jennifer"
Do : Sleep  : Until Me Is Nothing

Sub tv_Expand(Node As Node)
  If Node.Index <> 1 Then
    Node.Expanded = False    ' Prevent expand.
  EndIf
EndSub

Sub tv_Collapse(Node As Node)
  If Node.Index = 1 Then
```

```
    Node.Expanded = True    ' Marks it as still
      expanded but does not show children.
  EndIf
EndSub

Sub tv_NodeClick(Node As Node)
  If Node.Index = 1 Then
    Message "Node 1 Clicked"
  EndIf
EndSub
```

The above example causes some odd behaviour which can eventually result in branches disappearing altogether, so use with care.

See Also

TreeView, Expand, Style

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

NodeClick Event

Purpose

Occurs when a **Node** object is clicked.

Syntax

Sub *TreeView*_**NodeClick**(*Node* As Node)

Description

The **NodeClick** event passes a reference to the Node object that is clicked.

The standard Click event is generated when the user clicks any part of the **TreeView** control outside a node object. The NodeClick event is generated when the user clicks a particular **Node** object; the NodeClick event also returns a reference to a particular **Node** object which can be used to validate the **Node** before further action is taken.

The NodeClick event occurs before the standard Click event.

Example

```
Ocx TreeView tv1 = "", 10, 10, 150, 200
Local n
For n = 1 To 10 : tv1.Add , , Chr(64 + n) , "Node
 " & n : Next n
Do : Sleep : Until Me Is Nothing

Sub tv1_NodeClick(Node As Node)
```

```
  MsgBox "Index: " & Node.Index & #13#10 & "Key: "
    & Node.Key & #13#10 & "Text: " & Node.Text
EndSub
```

See Also

TreeView

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Arrange, View Property

Purpose

Arrange and **SnapToGrid** returns or sets a value that determines how the icons in a **ListView** control's Icon or SmallIcon **View** are arranged. **View** returns or sets the appearance of the **ListItem** objects in a **ListView** control.

Syntax

ListView.**Arrange** [= *value*]

ListView.**View** [= *value*]

value:iexp

Description

The **Arrange** settings for *value* are:

0	None	(Default)
1	Left	Items are aligned automatically along the left side of the control.
2	Top	Items are aligned automatically along the top of the control.

The **View** property values are:

0	Icon	(Default) Each **ListItem** object is represented by a full-sized (standard) icon and a text label.
1	SmallIcon	Each **ListItem** object is represented by a small icon and a text label that appears to

		the right of the icon. The items appear horizontally.
2	List	Each **ListItem** object is represented by a small icon and a text label that appears to the right of the icon. The **ListItem** objects are arranged vertically, each on its own line with information arranged in columns.
3	Report	Each **ListItem** object is displayed with its small icon and text labels. You can provide additional information about each **ListItem** object in a *subitem*. The icons, text labels, and information appear in columns with the leftmost column containing the small icon, followed by the text label. Additional columns display the text for each of the item's subitems.

Example

```
OpenW 1
' View property
Global Enum lvwIcon = 0, lvwSmallIcon, lvwList,
  lvwReport
' Arrange property (valid for lvwIcon,
  lvwSmallIcon)
Global Enum lvwNone = 0, lvwAutoLeft, lvwAutoTop
Ocx ImageList iml
iml.ListImages.Add , "comp",
  CreatePicture(LoadIcon(_INSTANCE, 1), False)
Ocx ListView lv = "", 100, 10, 140, 250, 200
lv.ColumnHeaders.Add , , "Column #1"
lv.ColumnHeaders.Add , , "Column #2"
lv.View = lvwSmallIcon
lv.Arrange = lvwAutoTop
lv.Icons = iml
lv.SmallIcons = iml
lv.Add , , "ListItem #1", "comp"
lv.ListItems.Add , , "ListItem #2", "comp"
lv.AddItem , , "ListItem #3", "comp"
Do
  Sleep
Until Me Is Nothing
```

See Also

ListView, Icons, SmallIcons

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

CheckBoxes, CheckedCount, CheckedItems Properties (ListView), Checked (ListItem)

Purpose

Returns or sets a value that determines if checkboxes appear, the number of boxes checked, and a collection of selected items.

Syntax

ListView.**CheckBoxes** [= Boolean]

ListView.**CheckedCount**

ListView.**CheckedItems**

ListItem.**Checked**

Description

When **CheckBoxes** = True, checkboxes will appear. By default, they don't appear.

The **CheckedCount** returns the number of checkboxes that are checked.

CheckedItems returns a reference to a **ListItems** collection containing all checked items.

ListItem.**Checked** returns or sets a Boolean that determines the checked state of the list item's check button.

Example

```
Local ch As ColumnHeader, li As ListItem
Ocx ListView lv = "", 10, 10, 300, 200
.View = 3 : .FullRowSelect = True
Set ch = lv.ColumnHeaders.Add( ,"1", "Checkbox") :
  ch.Width = 900
lv.ColumnHeaders.Add  , "2", "Description" :
  lv.ColumnHeaders(2).Width = 2500
lv.CheckBoxes = True
lv.Add , , ""  : lv.ListItem(1).AllText =
  ";Checkbox 1" : lv.ListItem(1).Checked = True
lv.Add , , ""  : lv.ListItem(2).AllText =
  ";Checkbox 2"
lv.Add , , ""  : lv.ListItem(3).AllText =
  ";Checkbox 3"
Do : Sleep : Until Me Is Nothing

Sub lv_ItemClick(Item As ListItem)
  lv_Report
EndSub

Sub lv_MouseUp(Button&, Shift&, x!, y!)
  lv_Report
EndSub

Sub lv_Report
  Local li As ListItem, a$
  If lv.CheckedItems.Count = 0 // lv.CheckedCount
    can be used instead
    a$ = "No CheckBoxes are checked."
  Else
```

```
    a$ = "The following items are
      checked:"#13#10#13#10
    For Each li In lv.CheckedItems
      a$ = a$ & "Checkbox " & li.Index & #13#10
    Next
  EndIf
  Message a$
EndSub
```

Remarks

CheckedCount is equivalent to **CheckedItems**.**Count**.

See Also

ListView, ListItems, ListItem

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Add Method (ColumnHeaders), DefaultWidth (ListView)

Purpose

Adds a **ColumnHeader** object to a **ColumnHeaders** collection.

The **DefaultWidth** property of the **ListView** parent control determines the default width of the column.

Syntax

*ListView.ColumnHeaders.***Add**([index], [key] , [caption])

*ListView.***DefaultWidth** [= value]

index, key, caption:Variant
value:Single

Description

The **ColumnHeaders**.**Add** method, adds or inserts a **ColumnHeader** object to the **ColumnHeaders** collection of the tool bar. The width of the column is preset with the Listview.**DefaultWidth** property.

index	Optional. An integer specifying the position where you want to insert the **ColumnHeader** object. If no *index* is specified, the ColumnHeader is added to the end of the **ColumnHeaders** collection.

key	Optional. A unique string that identifies the **ColumnHeader** object. Use this value to retrieve a specific **ColumnHeader** object.
caption	Optional. A string that will appear in the column header.

By default, the **DefaultWidth** property has the value 1440 Twips.

The width of the individual columns is modifies with **Width** property of the Column object returned from the **Add** method.

Example

```
' View property
Global Enum lvwIcon = 0, lvwSmallIcon, lvwList,
  lvwReport
Ocx ListView lv = "", 10, 10, 230, 200
lv.View = lvwReport
Dim col As ColumnHeader
lv.ColumnHeaders.Add , "1", "Column #1"
lv.ColumnHeaders.Add , "2", "Column #2"
Set col = lv.ColumnHeaders.Add( , "3", "Column
  #3")
col.Width = 2000
Do : Sleep : Until Me Is Nothing
```

See Also

ListView, ColumnHeaders, ColumnHeader

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

ExStyle Property (ListView)

Purpose

Returns or sets extended window styles for a **ListView** control.

Syntax

ListView.**ExStyle** [= value%]

Description

List view control styles have been extended. Not all styles have corresponding properties, but have to be set using the **ExStyle** property. This value can be a combination of LVS_EX_XXX style flags.

LVS_EX_CHECKBOXES Version 4.70. Enables check boxes for items in a list view control.

LVS_EX_FLATSB Version 4.71. Enables flat scroll bars in the list view. If you need more control over the appearance of the list view's scroll bars, you should manipulate the list view's scroll bars directly using the Flat Scroll Bar APIs.

LVS_EX_FULLROWSELECT Version 4.70. When an item is selected, the item and all its subitems are highlighted. This style is available only in conjunction with the LVS_REPORT style.

LVS_EX_GRIDLINES Version 4.70. Displays gridlines around items and subitems. This style is available only in conjunction with the LVS_REPORT style.

LVS_EX_HEADERDRAGDROP Version 4.70. Enables drag-and-drop reordering of columns in a list view control. This style is only available to list view controls that use the LVS_REPORT style.

LVS_EX_INFOTIP Version 4.71. The list view control sends an LVN_GETINFOTIP notification message to the parent window before displaying an item's tooltip. This style is only available to list view controls that use the LVS_ICON style.

LVS_EX_MULTIWORKAREAS Version 4.71. If the list view control has the LVS_AUTOARRANGE style, the control will not auto arrange its icons until one or more work areas are defined (see LVM_SETWORKAREAS). To be effective, this style must be set before any work areas are defined and any items have been added to the control.

LVS_EX_ONECLICKACTIVATE Version 4.70. The list view control sends an LVN_ITEMACTIVATE notification message to the parent window when the user clicks an item. This style also enables hot tracking in the list view control. Hot tracking means that when the cursor moves over an item, it is highlighted but not selected.

LVS_EX_REGIONAL Version 4.71. The list view will create a region that includes only the item icons and text and set its window region to that using SetWindowRgn. This will exclude any area that is not part of an item from the window region. This style is only available to list view controls that use the LVS_ICON style.

LVS_EX_SUBITEMIMAGES Version 4.70. Allows images to be displayed for subitems. This style is available only in conjunction with the LVS_REPORT style.

LVS_EX_TRACKSELECT Version 4.70. Enables hover selection in a list view control. Hover selection (also called track selection) means that an item is automatically selected when the cursor remains over the item for a certain period of time. The delay can be changed from the default system setting with the LVM_SETHOVERTIME message. This style applies to all styles of list view control.

LVS_EX_TWOCLICKACTIVATE Version 4.70. The list view control sends an LVN_ITEMACTIVATE notification message to the parent window when the user double-clicks an item. This style also enables hot tracking in the list view control. Hot tracking means that when the cursor moves over an item, it is highlighted but not selected.

LVS_EX_UNDERLINECOLD Version 4.71. Causes non-hot items to be displayed with underlined text. This style is ignored if LVS_EX_ONECLICKACTIVATE is not set.

LVS_EX_UNDERLINEHOT Version 4.71. Causes hot items to be displayed with underlined text. This style is ignored if LVS_EX_ONECLICKACTIVATE or LVS_EX_TWOCLICKACTIVATE is not set.

To fully exploit these styles see the MS SDK documentation.

Example

See Also

ListView

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

FullRowSelect, MultiSelect Properties, and SelectedCount, SelectedItems Methods (ListView)

Purpose

Returns or sets a value that determines if checkboxes appear, the number of boxes checked, and a collection of selected items.

Syntax

ListView.**FullRowSelect** [= Boolean]

ListView.**MultiSelect** [= Boolean]

ListView.**SelectedCount**

ListView.**SelectedItems**

object: ListView, TreeView

Description

FullRowSelect returns or sets a value that specifies if the entire row is selected. This property is only valid when the **View** property is set to *lvwReport* (3).

MultiSelect returns or sets a value indicating whether a user can select multiple objects or items. (Pressing SHIFT and clicking the mouse or pressing SHIFT and one of the arrow keys (UP ARROW, DOWN ARROW, LEFT ARROW, and RIGHT ARROW) extends the selection from the previously selected **ListItem** to the current **ListItem**. Pressing CTRL and clicking the mouse selects or deselects a **ListItem** in the list.)

The **SelectedCount** property returns the number of selected list items.

SelectedItems returns a reference to a **ListItems** collection containing all selected items.

Example

```
Global a$, m As Int, n As Int
Dim li As ListItem
Ocx ListView lv1 = , 10, 10, 500, 150 : lv1.View =
  3
Ocx Label lbl1 = "Selected Lines: 0", 10, 170,
  100, 14 : lbl1.BackColor = RGB(255, 255, 255)
Ocx Label lbl2 = "Selected Items:", 10, 185, 100,
  14 : lbl2.BackColor = RGB(255, 255, 255)
Ocx TextBox tx = "", 10, 200, 100, 200 :
  tx.BorderStyle = 1 : tx.MultiLine = True
For n = 1 To 5 : lv1.ColumnHeaders.Add , ,
  "Column" & n : Next n
For n = 1 To 5 :
  a$ = "" : For m = 1 To 5 : a$ = a$ & "Item " &
    ((n - 1) * 5) + m & Iif(m <> 5, ";", "") : Next
    m
  lv1.Add , , "" : lv1(n).AllText = a$
Next n
lv1.FullRowSelect = True
lv1.MultiSelect = True
```

```
Do : Sleep : Until Me Is Nothing

Sub lv1_Click
  lbl1.Caption = "Selected Lines: " &
    lv1.SelectedCount
  tx.Text = ""
  For Each li In lv1.SelectedItems
    a$ = li.AllText
    For n = 1 To 5 : m = InStr(a$, ";") : If m = 0
      Then m = Len(a$) + 1
      tx.Text = tx.Text & Left(a$, m - 1) & #13#10 :
        a$ = Mid(a$, m + 1)
    Next n
  Next
EndSub
```

Remarks

SelectedCount is equivalent to **SelectedItems.Count**.

SelectedItem is used in conjunction with **BeforeLabelEdit** event.

See Also

ListView, ListItems, ListItem

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Grid, GridLines Properties (ListView)

Purpose

Determine the use of a gridlines in a **ListView** control.

Syntax

% = *ListView*.**Grid**(*integer*)

ListView.**GridLines** [= Bool]

Description

Grid - No longer seems to enable gridlines in report view mode; the *integer* must be present and must be a value between 0 and 3; the return value can be ignored.

GridLines = True displays gridlines around items and subitems. This style is available only in conjunction with the *lvsReport* style (**View** = 3).

Example

```
Global a$, m As Int, n As Int
Dim li As ListItem
Ocx ListView lv1 = , 10, 10, 500, 150 : lv1.View =
  3
For n = 1 To 5 : lv1.ColumnHeaders.Add , ,
  "Column" & n : Next n
For n = 1 To 5 :
  a$ = "" : For m = 1 To 5 : a$ = a$ & "Item " &
    ((n - 1) * 5) + m & Iif(m <> 5, ";", "") : Next
```

```
   m
  lv1.Add , , "" : lv1(n).AllText = a$ : If n = 2
   Then lv1(n).Ghosted = True
Next n
lv1.FullRowSelect = True
lv1.GridLines = True
Do : Sleep : Until Me Is Nothing
```

Remarks

See Also

ListView, View

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Icons, SmallIcons Properties

Purpose

Returns or sets the **ImageList** controls associated with the *Icon* and *SmallIcon* views in a **ListView** control.

Syntax

[Set =] *ListView*.**Icons** [= imagelist]

[Set =] *ListView*.**SmallIcons** [= imagelist]

imagelist:ImageList Object

Description

To associate an **ImageList** control with a **ListView** control at run time, set these properties to the desired **ImageList** control.

Each **ListItem** object in the **ListView** control also has **Icon** and **SmallIcon** properties, which index the **ListImage** objects and determine which image is displayed.

Once you associate an **ImageList** with the ListView control, you can use the value of either the **Index** property to refer to a **ListImage** object in a procedure.

Example

```
Ocx ImageList iml
Ocx ListView lv = "", 100, 10, 140, 250, 200
```

```
iml.ListImages.Add , ,
 CreatePicture(LoadIcon(_INSTANCE, 1), False)
lv.Icons = iml
// or..
Set lv.SmallIcons = iml  // Set is optional
lv.Add , , "Icon", 1
Do : Sleep  : Until Me Is Nothing
```

See Also

ListView, ListItem

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

TextBackColor Properties (ListView)

Purpose

TextBackColor returns or sets a value that determines a **ListItem** object's text background color.

Syntax

ListView.**TextBackColor** [= value%]

Description

The *value*% argument specifies the background color of an object. The color value is a RGB value or a color constant **colxxx.** See BackColor.

When the **BackColor** property is adjusted, the **TextbackColor** is reset to the **BackColor** property value.

Example

```
Local n As Int32
Ocx ListView lv = , 10, 10, 200, 300
lv.TextBackColor = $c0c0
For n = 1 To 20 : lv.Add , , "Item" & n : Next n
Ocx Command cmd1 = "Change Back Color", 230, 10,
  140, 22
Ocx Command cmd2 = "Change Fore Color", 230, 40,
  140, 22
Do : Sleep : Until Me Is Nothing
```

```
Sub cmd1_Click
  Ocx CommDlg cd
  cd.ShowColor
  lv.TextBackColor = cd.Color
EndSub

Sub cmd2_Click
  Ocx CommDlg cd
  cd.ShowColor
  lv.ForeColor = cd.Color
EndSub
```

See Also

ListView

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

Sorted, TopIndex Properties (ComboBox, ListBox, ListView)

Purpose

Sorted returns a value indicating whether the elements of a **ComboBox** and **ListBox** are automatically sorted alphabetically.

TopIndex returns or sets a value that specifies which item in a **ComboBox, ListBox**, or a **ListView** control is displayed in the topmost position.

Syntax

*object***.Sorted** [**=** boolean]

*object***.TopIndex** [**=** *value*]

object:ListBox, ComboBox, ListView, TreeView

Description

ComboBox and **ListBox**

When **Sorted** is **True**, GFA-BASIC 32 handles almost all necessary string processing to maintain alphabetic order, including changing the index numbers for items as required by the addition or removal of items. Using the **InsertItem** method to add an element to a specific location in the list may violate the sort order, and subsequent additions may not be correctly sorted.

TopIndex = *value* sets the number of the list item that is displayed in the topmost position. The default is 0, or the first item in the list.

If the **Columns** property is set to 0 for the **ListBox** control, the item is displayed at the topmost position if there are enough items below it to fill the visible portion of the list. If the **Columns** property setting is greater than 0 for the **ListBox** control, the item's column moves to the leftmost position without changing its position within the column.

ListView

For a **ListView** control the **TopIndex** property is read-only and returns the number of the topmost **ListItem**.

Example

```
Global Int32 n
AutoRedraw = 1
Ocx ListBox lb = "", 10, 10, 100, 200 : lb.Sorted
  = False
For n = 40 DownTo 1 : lb.AddItem "Item " &
  Format(n, "00") : Next n
Ocx TextBox tb = "", 120, 10, 40, 14 :
  .BorderStyle = 1 : .ReadOnly = True : Text 170,
  11, "ListBox TopIndex Value"
Ocx UpDown up : .BuddyControl = tb : .Min = 0 :
  .Max = 40 : .Increment = 1 : .Value = lb.TopIndex
Ocx CheckBox chk = "Sort ListBox Entries", 120,
  30, 120, 14
Do : Sleep : Until Me Is Nothing

Sub chk_Click
  Local ti As Int32 = lb.TopIndex
  lb.Sorted = -chk.Value
  For n = 40 DownTo 1 : lb.AddItem "Item " &
    Format(n, "00") : Next n
  lb.TopIndex = ti
EndSub

Sub up_Change
  lb.TopIndex = up.Value
EndSub
```

See Also

ListBox, ComboBox, ListView, ListItem

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Add, AddItem Method (ListView, ListItems)

Purpose

Adds an **ListItem** to a **ListItems** collection in a **ListView** control and returns a reference to the newly created **ListItem** object.

Syntax

*ListView***.Add[Item]**([index], [key], [text], [icon] [, smallicon])

*ListItems***.Add**([index], [key], [text], [icon] [, smallicon])

object:ListView, ListItems
Index, Key, Text, Icon, smallicon: Variant exp

Description

The **ListView** Ocx has the **AddItem** and **Add** methods, which act exactly the same. The **ListItems** object supports the **Add** method only.

index	Optional. An integer specifying the position where you want to insert the **ListItem**. If no index is specified, the **ListItem** is added to the end of the **ListItems** collection.
key	Optional. A unique string expression that can be used to access a member of the collection.
text	Optional. A string that is associated with the **ListItem** object control.

icon	Optional. An integer that sets the icon to be displayed from an ImageList control, when the **ListView** control is set to Icon view.
smallicon	Optional. An integer that sets the icon to be displayed from an **ImageList** control, when the **ListView** control is set to SmallIcon view.

Before setting either the **Icons** or **SmallIcons** properties, you must first initialize them. You can do this at design time by specifying an **ImageList** object, or at run time with the following code:

```
listview1.Icons = iml1          'Assuming the
  Imagelist is iml1.
listview1.SmallIcons = iml2
```

If the list is not currently sorted, a **ListItem** object can be inserted in any position by using the *index* argument. If the list is sorted, the *index* argument is ignored and the **ListItem** object is inserted in the appropriate position based upon the sort order.

If *index* is not supplied, the **ListItem** object is added with an index that is equal to the number of **ListItem** objects in the collection + 1.

Use the **Key** property to reference a member of the **ListItems** collection if you expect the value of an object's **Index** property to change, such as by dynamically adding objects to or removing objects from the collection.

Example

```
Dim li As ListItem
OpenW 1, 20, 20 , 300, 300
Ocx ListView lvw = , 10, 10, 100, 100
```

```
lvw.View = 2
lvw.AddItem 1, , "First"
lvw.Add 2, , "Second"
lvw.ListItems.Add 3, , "Third"
Set li = lvw.Add( , , "Fourth")
Do
  Sleep
Until Me Is Nothing
```

Remarks

GFA-BASIC 32 specific

Instead of explicitly using the **ListItems** collection to access a **ListItem** element, you can use a shorter notation. First, the **ListView** supports an **Item** property:

```
lvw.Item(idx), lvw.ListItems.Item(idx)
```

Like the **Item** method of lvw.**ListItems, Item** is the default method of **ListView**. Therefore, a **ListItem** can be accessed as follows:

```
lvw(idx) , lvw.ListItems(idx)
```

```
lvw!idx, lvw.ListItems!idx
```

Each dot saves about 30 bytes of code.

To enumerate over the **ListItems** collection of a **ListView** Ocx, use For Each on the Ocx control directly, like:

```
Local li As ListItem
For Each li In lvw : DoSomething(li) : Next
```

See Also

ListView, ListItem, ListItems

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

AddItem, InsertItem, RemoveItem, Clear Method, NewIndex Property

Purpose

ListBox or **ComboBox**: **AddItem** adds an item. **RemoveItem** removes an item **NewIndex** returns the index of the item most recently added.

Syntax

*object***.AddItem** item$ [, itemdata%]

*object***.InsertItem** index%**,** item$ [, itemdata]

*object***.RemoveItem** index%

*object***.Clear**

*object***.NewIndex**

object:ListBox, ComboBox
item$:sexp
itemdata:iexp

Description

AddItem adds an item to the **ListBox** or **ComboBox**. It takes the text for the item and optional the *itemdata*, a long integer value linked to the item. **InsertItem** inserts an item at the specified position. Both **AddItem** and **InsertItem** return the actual position of the list item.

NewIndex returns the index of most recently added. You can use the **NewIndex** property with sorted lists when you need a list of values that correspond to each item in the **ItemData** property array. As you add an item in a sorted list, GFA-BASIC 32 inserts the item in the list in alphabetic order. This property tells you where the item was inserted so that you can insert a corresponding value in the **ItemData** property at the same index.

RemoveItem removes a specified item from the list. **Clear** removes all items from the **ListBox** or **ComboBox**.

Example

```
Dim i As Int
OpenW 1, 20, 20 , 300, 300
Ocx ListBox lb1 = , 10, 10, 100, 100
For i = 1 To 100   ' Count from 1 to 100.
  lb1.AddItem Dec(i, 3) & "Entry "
Next
MsgBox "Choose OK to remove every other entry."
For i = 1 To 50
  lb1.RemoveItem i
Next
MsgBox "Choose OK to remove all items from the
  list box."
lb1.Clear   ' Clear list box.
Do
  Sleep
Until Me Is Nothing
```

Remarks

If you supply a valid value for *index*, *item* is placed at that position within the *object*. If *index* is omitted, *item* is added at the proper sorted position (if the **Sorted** property is set to **True**) or to the end of the list (if **Sorted** is set to **False**).

See Also

ListBox, ComboBox

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

LineItem, ListItem, Item, GetFirstVisible Methods (ListView)

Purpose

These **ListView** methods return a reference to a **ListItem** object.

Syntax

ListView.**LineItem**(index%)

ListView.**ListItem**(index%)

ListView.**Item**(variant)

ListView.**GetFirstVisible**

Description

LineItem and **ListItem** return a reference to the specified index of the list item object (index starts with 1).

Item returns an item from the **ListItems** collection of the **ListView** control by either name or index.

GetFirstVisible returns a reference to the first object visible in the internal area of a control. A **ListView** control can contain more **ListItem** objects than can be seen in the internal area of the **ListView** control. You can use the reference returned by the **GetFirstVisible** method to determine the first visible **ListItem** object in List or Report view.

Example

```
Debug.Show
~SetWindowPos(Debug.hWnd, 0, 205, 10, 600, 500, 0)
OpenW 1, 10, 10, 185, 300
Local n As Int32
Dim li As ListItem
Ocx ListView lv = "", 10, 10, 150, 200 : lv.View =
  3
lv.ColumnHeaders.Add , , "Column1" :
  lv.ColumnHeaders(1).Width = PixelsToTwipX(130)
For n = 1 To 20
  lv.ListItems.Add , "p" & Trim(n) , "Item " & n
  Set li = lv.ListItem(n)
  If Odd(n) Then li.Bold = True Else li.Italic =
    True
Next n
Ocx Timer lv_tim : lv_tim.Interval = 100 :
  lv_tim.Enabled = True
For n = 1 To 20 Step 5
  // All the following produce the same result
  Trace lv.LineItem(n).Text
  Trace lv.ListItem(n + 1).Text
  Trace lv.ListItems.Item("p" & Trim(n + 2)).Text
    // Item can take the Key string
  Trace lv.Item(n + 3).Text
    // ...or the Index number
  Trace lv(n + 4).Text
Next n
Do : Sleep  : Until Me Is Nothing
Debug.Hide

Sub lv_tim_Timer
```

```
  // In place of a 'Scroll' event which ListView is
    missing
  Static Int32 sp : Local Int32 nsp
  nsp = GetScrollPos(lv.hWnd, SBS_VERT)
  If sp <> nsp
    sp = nsp
    Set li = lv.GetFirstVisible // - If this
      doesn't work see Known Issues below
    ' Const LVM_GETTOPINDEX = 4135
    ' Set li = lv(SendMessage(lv.hWnd,
      LVM_GETTOPINDEX, 0, 0) + 1)
    Debug "Top Item: ";li.Text
  EndIf
EndSub
```

Known Issues

Prior to OCX v2.33/2.34, **GetFirstVisible** returned **Nothing** (bug). If you experience this problem, you should download the latest version of GfaWin23.ocx; otherwise, use LVM_GETTOPINDEX instead as follows:

```
Const LVM_GETTOPINDEX = 4135
Set li = lv(SendMessage(lv.hWnd, LVM_GETTOPINDEX,
  0, 0) + 1)
```

See Also

ListView, ListItem

{Created by Sjouke Hamstra; Last updated: 04/03/2018 by James Gaite}

SetGrid, SnapToGrid Method (ListView)

Purpose

Moves an item to a specified position in a list view control (must be in icon or small icon view).

Syntax

ListView.**SetGrid**(index, x, y)

ListView.**SnapToGrid**

index, x, y:iexp

Description

The *index* parameter specifies the index of the list view item, which should be an Icon or SmallIcon. The *x* and *y* parameters specify the new position of the item's upper-left corner, in view coordinates.

If the **Arrange** property = 0 (no auto arrange) the items in the list view control are arranged when invoking **SnapToGrid**.

The **SnapToGrid** method snaps all icons to the nearest grid position.

Example

```
Local Int32 n
```

```
Ocx ImageList iml : .ImageHeight = 32 :
  .ImageWidth = 32
For n = 32512 To 32518 : iml.Add , "Img " & n,
  CreatePicture(LoadIcon(Null, n)) : Next n
Ocx ListView lv = "", 10, 10, 400, 300 : .Icons =
  iml : .SmallIcons = iml : .Arrange = 1
For n = 1 To 7 : lv.Add , , "Icon" & n, n, n :
  Next n
~lv.SetGrid(2, 90, 100)
lv.SnapToGrid
Do : Sleep : Until Me Is Nothing
```

Remarks

Rather than just influencing the position of one icon, **SetGrid** seems to affect the positioning of all.

See Also

ListView, Arrange

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Sort Method (ListView)

Purpose

Sorts the items in a ListItems collection.

Syntax

ListView.**Sort** column%, compare%

Description

Sorts the **ListView** control using the specified *column*% as sort key. The value for *column*% starts with 0, where the **ColumnHeader** indices start with 1.

It is common to sort a list when the column header is clicked. For this reason, the **Sort** property is commonly included in the **ColumnClick** event to sort the list using the clicked column. It is also common to sort the list in ascending order first and when the column header is clicked again in descending order.

compare% specifies a value that determines whether **ListItem** objects in a **ListView** control are sorted in ascending or descending order and which compare mode to use. The value for *compare*% is the same as for **Mode Compare** (0 = Binary, 1 = Text - case insensitive, etc). The descending sort order is specified by setting the appropriate bit in the *compare*% value ($10000)

$0 - Ascending order. Sorts from the beginning of the alphabet (A-Z) or the earliest date. Numbers are sorted as strings, with the first digit determining the initial position in the sort, and subsequent digits determining sub-sorting.

$10000 - Descending order. Sorts from the end of the alphabet (Z-A) or the latest date. Numbers are sorted as strings, with the first digit determining the initial position in the sort, and subsequent digits determining sub-sorting.

Example

```
Global a$, n As Int32
Ocx ListView lv = "", 10, 10, 400, 200 : .View = 3
  : .FullRowSelect = True : .GridLines = True
For n = 1 To 4 : lv.ColumnHeaders.Add , , "Column"
  & n : lv.ColumnHeaders(n).Alignment = 2 : Next n
For n = 1 To 4
  lv.Add , , ""
  a$ = Rand(10) & ";" & Rand(10) & ";" & Rand(10) &
    ";" & Rand(10)
  lv(n).AllText = a$
Next n
Do : Sleep : Until Me Is Nothing

Sub lv_ColumnClick(ColumnHeader As ColumnHeader)
  // ColumnHeader objects do not store their sort
    direction...
  // ...so you can use Tag instead
  If Val(ColumnHeader.Tag) = 0 Or
    Val(ColumnHeader.Tag) = 1
    lv.Sort ColumnHeader.Index - 1, $1
    ColumnHeader.Tag = 2
  Else
    lv.Sort ColumnHeader.Index - 1, $10000
    ColumnHeader.Tag = 1
  EndIf
EndSub
```

The **ListView** lv1 is sorted case-insensitive (compare% = 1) and the sort order is toggled when the column is clicked again.

Remarks

For the possible values for **Mode Compare** see here.

See Also

ListView, ColumnClick

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

ColumnClick, ItemClick Events (ListView)

Purpose

ColumnClick occurs when a **ColumnHeader** object in a **ListView** control is clicked. Only available in Report view (3).

ItemClick occurs when a **ListItem** object in a **ListView** control is clicked.

Syntax

Sub *ListView*_**ColumnClick**(ColumnHeader As ColumnHeader)

Sub *ListView*_**ItemClick**(Item As ListItem)

Description

The **ColumnClick**(ColumnHeader As ColumnHeader) event commonly use the **Sort** property to sort the **ListItem** objects in the clicked column.

Use the **ItemClick**(Item As ListItem) event to determine which **ListItem** was clicked. This event is triggered before the **Click** event. The standard **Click** event is generated if the mouse is clicked on any part of the **ListView** control. The **ItemClick** event is generated only when the mouse is clicked on the text or image of a **ListItem** object.

Example

```
Ocx ListView lv1 = "", 10, 10, 200, 200
.View = 3 : .FullRowSelect = True
lv1.ColumnHeaders.Add  , , "Column1" :
  lv1.ColumnHeaders.Add , , "Column2"
lv1.Add , , "" : lv1.ListItem(1).AllText =
  "Bobby;Moore"
lv1.Add , , "" : lv1.ListItem(2).AllText =
  "Jack;Charlton"
lv1.Add , , "" : lv1.ListItem(3).AllText =
  "Bobby;Charlton"
Do : Sleep  : Until Me Is Nothing

Sub lv1_ItemClick(Item As ListItem)
  Message(Item.SubItems(0) & " " &
    Item.SubItems(1))
EndSub

Sub lv1_ColumnClick(ColumnHeader As ColumnHeader)
  Global lv1IsSorted As Int
  Const lvwDescending = $10000
  If ColumnHeader.Index == lv1IsSorted
    lv1.Sort lv1IsSorted - 1, 1 + lvwDescending
    lv1IsSorted = 0
  Else
    lv1IsSorted = ColumnHeader.Index
    lv1.Sort lv1IsSorted - 1, 1
  EndIf
End Sub
```

The **ListView** lv1 is sorted case-insensitive (compare% = 1) and the sort order is toggled when the column is clicked again.

To recognize a second click a global variable IsSorted is used (here lv1IsSorted to identify the control). The IsSorted variable holds the latest clicked column. When a column is clicked for a second time lv1IsSorted is equal to the Index of the column. But, of course the **Tag** property of the **ColumnHeader** item can also be used to store the current sort order.

See Also

ListView, ColumnHeader, ListItem, Sort

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Max, Min, LargeChange, SmallChange, Value Properties

Purpose

Returns or set the minimum, maximum, and the amount of change to the **Value** property.

Syntax

*object***.Max** [**=** value%]

*object***.Min** [**=** value%]

*object***.Value** [= value%]

objects.**LargeChange** [= value%]

objects.**SmallChange** [= value%]

object:Scroll, Slider, ProgressBar Ocx
objects:Scroll, Slider Ocx

Description

Min and **Max** return or set the minimum and maximum for the **Value** property for the specified control. The **Value** property returns or sets the current position control. For each property, you can specify an integer between -32,768 and 32,767, inclusive. For all Ocx control types the default setting for **Min** is 0. For a **Scroll** control **Max** = 1000, for a

Slider and **ProgressBar** control **Max** = 100. But these can be changed.

Slider - The **LargeChange** property sets the number of ticks the slider will move when you press the PAGEUP or PAGEDOWN keys, or when you click the mouse to the left or right of the slider, default = 20. The **SmallChange** property sets the number of ticks the slider will move when you press the left or right arrow keys, default = 1.

Scroll - The **LargeChange** returns or sets the amount of change to the **Value** property setting in a scroll bar control when the user clicks the area between the scroll box and scroll arrow, default = 100. The **SmallChange** returns or sets the amount of change to the **Value** property setting in a scroll bar control when the user clicks a scroll arrow, default = 1.

Example

```
OpenW Center # 1, , , 400, 200
Me.BackColor = colBtnFace
Ocx Scroll sc1 = "", 10, 10, 370, 20
Ocx ProgressBar pb1 = "", 10, 50, 370, 20
With sc1
  .Min = 0 : .Max = 600
  .LargeChange = (.Max - .Min) / 10 : .SmallChange
    = 10
End With
Do
  Sleep
Loop Until Me Is Nothing

Sub sc1_Scroll()
  pb1.Value = (sc1.TrackValue * 10 / 9) / ((sc1.Max
    - sc1.Min) / 100)
```

```
Sub sc1_Change()
 pb1.Value = (sc1.Value * 10 / 9) / ((sc1.Max -
   sc1.Min) / 100)
```

See Also

Scroll, Slider, ProgressBar

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

Scroll, Change Events, TrackValue Property (Slider, Scroll)

Purpose

Occurs when you move the slider on a **Slider** control or the scroll box on a **Scroll** control, either by clicking on the control or using keyboard commands.

Syntax

Sub *object*_**Scroll(** [*index%*] **)**

Sub *object*_**Change(** [*index%*]**)**

*Scroll***.TrackValue** [= *value*]

object:Scroll, Slider Ocx
index%:iexp, index when control is part of control array
value:iexp

Description

The **Change** event indicates the contents of a control have changed. In fact, the **Change** event is triggered when the **Value** property has changed. The event is triggered after the **Scroll** or **Slider** control has changed. By contrast, the **Scroll** event is continually triggered during the dragging. During the **Scroll** event the value of the scroll box can be obtained using the **TrackValue** property. The **TrackValue** property is a **Scroll** Ocx property valid only in the **Scroll** event sub, and is not shared with the **Slider** control. The current value for the **Slider** is simply obtained using **Value**.

A **Scroll** event is always followed by a **Change** event.

Example

```
OpenW Center # 1, , , 400, 200
Me.BackColor = colBtnFace
Ocx Scroll sc1 = "", 10, 10, 370, 20
Ocx Label lb0 = "Value:", 10, 50, 100, 20
.Alignment = basRightJustify
Ocx Label lb1 = Str(sc1.Value), 120, 50, 50, 20
Ocx Slider sl1 = "", 10, 90, 370, 20
Ocx Label lb01 = "Value:", 10, 130, 100, 20
.Alignment = basRightJustify
Ocx Label lb2 = Str(sl1.Value), 120, 130, 50, 20
Do
  Sleep
Loop Until Me Is Nothing

Sub sc1_Scroll()
  lb1.Text = Str$(sc1.TrackValue)
EndSub

Sub sc1_Change()
  lb1.Text = Str$(sc1.Value)
EndSub

Sub sl1_Scroll
  lb2.Text = Str$(sl1.Value)
EndSub

Sub sl1_Change
  lb2.Text = Str$(sl1.Value)
EndSub
```

Remarks

The **Scroll** and **Change** events can be compared to the Form Scrollbars as follows:

_HScrolling, _VScrolling events_**Scroll** event

HScTrack, VScTrack properties**TrackValue** property

_HScroll, _VScroll events _**Change** event

HScPos, VScPos properties **Value** property (Default)

See Also

Scroll, Slider

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

SelectRange, SelLength, SelStart Properties, ClearSel (Slider Ocx)

Purpose

Returns or set the selection state of a **Slider** control.

Syntax

Slider.**SelectRange** = *boolean*

Slider.**SelLength** [= *value%*]

Slider.**SelStart** [= *value%*]

Slider**.ClearSel**

Description

SelectRange determines whether or not the **Slider** can have a selected range.

SelLength returns or sets the length of a selected range, and **SelStart** returns or sets the start of a selected range in a **Slider** control. The value falls within the **Min** and **Max** properties.

The **SelLength** and **SelStart** properties are used together to select a range of contiguous values on a **Slider** control. The **Slider** control then has the additional advantage of being a visual analog of the range of possible values.

The **SelLength** property can't be less than 0, and the sum of **SelLength** and **SelStart** can't be greater than the **Max** property.

If **SelectRange** is set to **False**, then the **SelStart** property setting is the same as the **Value** property setting. Setting the **SelStart** property also changes the **Value** property, and vice-versa, which will be reflected in the position of the slider on the control. Setting **SelLength** when the **SelectRange** property is **False** has no effect.

The **ClearSel** method cears the current selection range in a slider.

Example

```
Global sldclr?
Ocx Slider sld = "", 10, 10, 400, 20 : .Min = 10 :
  .Max = 200 : .TickFrequency = 10 : .LargeChange =
  10
Ocx CheckBox chk = "Allow Range Selection", 10,
  40, 130, 14
sld_Scroll
Do : Sleep : Until Me Is Nothing

Sub chk_Click
  sld.SelectRange = -chk.Value
EndSub

Sub sld_Change
  sldclr? = True
EndSub

Sub sld_MouseDown(Button&, Shift&, x!, y!)
  If sldclr? Then sld.ClearSel : sldclr? = False
EndSub
```

```
Sub sld_Scroll
  Local fn$ = FontName : FontName = "courier new"
  If sld.SelectRange And sld.SelLength > 0
    Text 10, 60, "Slider Start Position: " &
      sld.SelStart & Space(2)
    Text 10, 75, "Slider Range Length: " &
      sld.SelLength & Space(2)
  Else
    Text 10, 60, "Slider Position: " & sld.Value &
      Space(20)
    Text 10, 75, Space(40)
  EndIf
  FontName = fn$
EndSub
```

See Also

Slider

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

TickFrequency, TickStyle Properties, GetNumTicks Method

Purpose

Returns or sets (the frequency of) tick marks on a **Slider** control.

Syntax

object.**TickFrequency** [= *number%*]

object.**GetNumTicks**

object.**TickStyle** [= *number%*]

object:Slider Ocx

Description

TickFrequency returns or sets the frequency of tick marks on a **Slider** control in relation to its range. For example, if the range is 100, and the **TickFrequency** property is set to 2, there will be one tick for every 2 increments in the range. To change the number of ticks, reset the **Min** or **Max** properties or the **TickFrequency** property.

GetNumTicks returns the number of ticks between the **Min** and **Max** properties.

TickStyle returns or sets the style (or positioning) of the tick marks displayed on the **Slider** control.

TickStyle = 0 - ticks at the bottom or at the right
TickStyle = 1 - ticks at the top or at the left
TickStyle = 2 - ticks at the both sides
TickStyle = 3 - none. no ticks

Example

```
Local a$, n%
OpenW Center # 1, , , 400, 200
Me.BackColor = colBtnFace
Ocx Slider sli1 = "", 0, 0, 350, 45
.TickStyle = 0
.TickFrequency = 20
.Appearance = 3
Text 10, 50, "Minimum Value:" : Ocx TextBox tb(1)
  = "", 90, 49, 45, 15 : tb(1).BorderStyle = 1 :
  tb(1).Text = sli1.Min
Text 10, 68, "Maximum Value:" : Ocx TextBox tb(2)
  = "", 90, 67, 45, 15 : tb(2).BorderStyle = 1 :
  tb(2).Text = sli1.Max
Text 10, 86, "TickFrequency:" : Ocx TextBox tb(3)
  = "", 90, 85, 45, 15 : tb(3).BorderStyle = 1 :
  tb(3).Text = sli1.TickFrequency
Text 10, 104, "Tick Style:" : Ocx ComboBox cmb =
  "", 90, 101, 120, 22 : cmb.Style = 2
For n% = 0 To 3 : Read a$ : cmb.AddItem a$, n% :
  Next n% : cmb.ListIndex = 3
Data Ticks Top/Left,Ticks Bottom/Right,Ticks Both
  Sides,No Ticks
Do
  Sleep
Loop Until Me Is Nothing

Sub cmb_Click
  sli1.TickStyle = cmb.ItemData(cmb.ListIndex)
EndSub
```

```
Sub tb_Change(Index%)
  Select Index%
  Case 1 : sli1.Min = Val(tb(1).Text)
  Case 2 : sli1.Max = Val(tb(2).Text)
  Case 3 : sli1.TickFrequency = Val(tb(3).Text)
  EndSelect
EndSub
```

See Also

Slider

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Add, AddItem Method (ToolBar, Buttons)

Purpose

Add a **Button** object to a **Buttons** collection.

Syntax

*ToolBar.***Add**[**Item**] ([index], [key] , [caption], [style], [image])

*Buttons.***Add**([index], [key] , [caption], [style], [image])

index, key, caption, style, image:Variant

Description

The **ToolBar** methods **Add** and **AddItem,** and **Buttons**.**Add** method, add or insert a **Button** object to the **Buttons** collection of the tool bar.

index	Optional. An integer specifying the position where you want to insert the **Button** object. If no *index* is specified, the **Button** is added to the end of the **Buttons** collection.
key	Optional. A unique string that identifies the **Button** object. Use this value to retrieve a specific **Button** object.
caption	Optional. A string that will appear beneath the **Button** object.
style	Optional. The style of the **Button** object. 0Default. The button is a regular push button.

	1Checked. The button is a check button, which can be checked or unchecked. 2Button group. The button remains pressed until another button in the group is pressed. Exactly one button in the group can be pressed at any one moment. 3Separator. The button functions as a separator with a fixed width of 8 pixels. 4Place Holder. The button is like a separator in appearance and functionality, but has a settable width.
image	Optional. An integer or unique key that specifies a **ListImage** object in an associated **ImageList** control.

Buttons that have the Button Group (2) style must be grouped. To distinguish a group, place all **Button** objects with the same style (Button Group) between two **Button** objects with the Separator (3) style.

When a **Button** object is assigned the Place Holder (4) style, you can set the value of the **Width** property to accommodate another control placed on the **Button**. If a **Button** object has the Button, Check, or Button Group style, the height and width are determined by the **Height** and **Width** properties.

Example

```
AutoRedraw = 1
Ocx ToolBar tb
Dim btn As Button
Set btn = tb.Buttons.Add( , "open", "Open" , 0)
tb.Add , , , 3   ' Separator
tb.AddItem 1, , "New" , 0
```

```
OcxOcx tb ComboBox cb = , tb.Button(3).Left, 0,
  100, 1
' The height of the ComboBox depends on the Font.
cb.Top = -cb.Height - 3
Do : Sleep : Until Me Is Nothing
```

Remarks

GFA-BASIC 32 specific

Instead of explicitly using the **Buttons** collection to access a **Button** element, you can use a shorter notation. First, the **ToolBar** Ocx supports an **Item** property:

tb.**Item**(idx)tb.**Buttons**.**Item**(idx)

Like the **Item** method of tb.**Buttons, Item** is the default method of **ToolBar**. Therefore, a **Button** can be accessed as follows:

tb(idx)tb.**Buttons**(idx)

tb!idxtb.**Buttons**!idx

Each dot saves about 30 bytes of code.

To enumerate over the **Buttons** collection of a **ToolBar** Ocx, use For Each on the Ocx control directly, like:

```
Local btn As Button
For Each btn In tb : DoSomething(btn) : Next
```

See Also

ToolBar, Buttons

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

ButtonClick, ButtonDblClick Events

Purpose

Occurs when the user clicks or double clicks on a **Button** object in a **Toolbar** control.

Syntax

Sub *object*_**ButtonClick**(*btn* **As Button**)

Sub *object*_**ButtonDblClick**(*btn* **As Button**)

object: ToolBar Ocx

Description

To program an individual **Button** object's response to the ButtonClick event, use the value of the *btn* argument. For example, use the **Key** property of the **Button** object to determine the appropriate action.

Example

```
Ocx ToolBar tlb
tlb.Add , "open" , "Open"
tlb.Add , "save" , "Save"
Do : Sleep : Until Me Is Nothing

Sub tlb_ButtonClick(Btn As Button)
  Switch Btn.Index
  Case 1 : Message("Open Button") // open
  Case 2 : Message("Save Button") // save
  EndSwitch
EndSub

Sub tlb_ButtonDblClick(Btn As Button)
  Trace Btn
  Switch Btn.Key
  Case "open" : Message("Open DClick")
  Case "save" : Message("Save DClick")
  EndSwitch
EndSub
```

Remarks

When afterwards a **Button** is inserted, the **Index** property might lose its original value. Therefore, it is often better to use the **Key** property, which uniquely identifies a button.

See Also

ToolBar

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

SimpleText, Style Properties (StatusBar)

Purpose

Control the style and text of the **StatusBar** control.

Syntax

StatusBar.**SimpleText** [= string]

StatusBar.**Style** [= number]

Description

The **Style** property returns or sets the style of a **StatusBar** control. When Style = 0 the status bar shows all panels, but when Style = 1 the control display only one large panel.

Use the **SimpleText** property to set the text of the string to be displayed when the **Style** property is set to 1. **NOTE** giving a value to **SimpleText** automatically sets **Style** = 1.

Example

```
Ocx StatusBar sb : .SimpleText = "Style is 1" :
  .Style = 0
sb.Add , , "Panel 1"
sb.Add , , "Style is 0"
sb.Add , , "CAPS" , 3
Ocx CheckBox chk = "Change Style to 1", 10, 10,
  120, 14
Do : Sleep : Until Me Is Nothing
```

```
Sub chk_Click
  sb.Style = chk.Value
EndSub
```

See Also

StatusBar

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Add, AddItem Method (StatusBar, Panels)

Purpose

Add a **Panel** object to a **Panels** collection.

Syntax

*StatusBar.***Add**[**Item**] ([index], [key], [caption], [style], [picture])

*StausBar.Panels.***Add**([index], [key], [caption], [style], [picture])

index, key, caption, style, image:Variant

Description

The **StatusBar** methods **Add** and **AddItem,** and the **Panels**.**Add** method, add or insert a **Panel** object to the **Panels** collection of the status bar.

index	Optional. An integer specifying the position where you want to insert the **Panel** object. If no *index* is specified, the **Panel** is added to the end of the **Panels** collection.
key	Optional. A unique string that identifies the **Panel** object. Use this value to retrieve a specific **Panel** object.
caption	Optional. A string that will appear in the **Panel** object.
style	Optional. The style of the **Panel** object.

	0 - Default. Text and/or a bitmap. Set text with the **Text** property. The **Panel** appears to be sunk into the status bar.
	1 - Flat. The **Panel** displays no bevel, and text looks like it is displayed right on the status bar.
	2 - Raised. The **Panel** appears to be raised above the status bar.
	3 - Caps Lock key. Displays the letters CAPS in bold when Caps Lock is enabled, and dimmed when disabled.
	4 - Number Lock. Displays the letters NUM in bold when the number lock key is enabled, and dimmed when disabled.
	5 - Scroll Lock key. Displays the letters SCRL in bold when scroll lock is enabled, and dimmed when disabled.
	6 - Insert key. Displays the letters INS in bold when the insert key is enabled, and dimmed when disabled.
	7 - Date. Displays the current date and time in the system format or in the format specified in *caption*. The custom format used has to be the same as specified in **Format**().
image	Optional. An integer or unique key that specifies a **ListImage** object in an associated **ImageList** control.

GFA-BASIC 32 omits the *Style* constants for a **Panel**, instead you could use the following Enum.

Example

```
' Constants for the StatusBar Panel Styles
```

```
Global Enum sbrText = 0, sbrFlat, sbrRaise,
  sbrCaps, sbrNum, sbrScroll, sbrIns, sbrDate
' Constants for Text alignment in StatusBar Panels
Global Enum sbrLeft = 0, sbrCenter, sbrRight
Ocx StatusBar sb
sb.Panels.Add , , "Part 1", sbrText
sb.Panels.Add , , "Part 2", sbrFlat
sb.Panels.Add , , "Part 3", sbrRaise
sb.Panels.Add , , "Caps", sbrCaps
sb.Panels.Add , , "Num", sbrNum
sb.Panels.Add , , "Scroll", sbrScroll
sb.Panels.Add , , "INS", sbrIns
sb.Panels.Add , , "c", sbrDate
Do : Sleep : Until Me Is Nothing
```

Remarks

GFA-BASIC 32 specific

Instead of explicitly using the **Panels** collection to access a **Panel** element, you can use a shorter notation. First, the **StatusBar** supports an **Item** property:

sb.**Item**(idx)sb.**Panels**.**Item**(idx)

Like the **Item** method of sb.**Panels, Item** is the default method of **StatusBar**. Therefore, a **Panel** can be accessed as follows:

sb(idx)sb.**Panels**(idx)

sb!idxsb.**Panels**!idx

Each dot saves about 30 bytes of code.

To enumerate over the **Panels** collection of a **StatusBar** Ocx, use For Each on the Ocx control directly, like:

```
Local p As Panel
For Each p In sb : DoSomething(p) : Next
```

Known Issues

The *index* property of **Add[Item]** works inconsistently: most of the time, it will do as desired, but sometimes the panel position will not be affected it by it and, on others, an 'Illegal Function Call' error is thrown.

The following example will throw the 'Illegal Function Call' error; this usually only happens if the index is the same as the panel number being created, below one or greater than the number of panels:

```
Ocx StatusBar sb
sb.AddItem , , "Panel 1"
sb.AddItem 2, , "Panel 2"
sb.AddItem , , "Panel 3"
Do : Sleep : Until Me Is Nothing
```

The following example, trying to set Panel 2 to be the first panel just does nothing...

```
Ocx StatusBar sb
sb.AddItem , , "Panel 1"
sb.AddItem 1, , "Panel 2"
sb.AddItem , , "Panel 3"
Do : Sleep : Until Me Is Nothing
```

...while this one setting Panel 3 to the first position does work.

```
Ocx StatusBar sb
sb.AddItem , , "Panel 1"
sb.AddItem , , "Panel 2"
sb.AddItem 1, , "Panel 3"
```

```
Do : Sleep : Until Me Is Nothing
```

There is no workaround for this and the best advice is to use caution when setting the *index* parameter.
[Reported by Jean-Marie Melanson, 26/02/2017]

See Also

StatusBar, Panels

{Created by Sjouke Hamstra; Last updated: 02/03/2017 by James Gaite}

PanelClick, PanelDblClick Events

Purpose

Occurs when the user clicks or double clicks on a **Panel** object in a **StatusBar** control.

Syntax

Sub *object*_**PanelClick**(*p* **As Panel**)

Sub *object*_**PanelDblClick**(*p* **As Panel**)

object: StatusBar Ocx

Description

The **PanelClick** event is similar to the standard Click event but occurs when a user presses and then releases a mouse button over any of the **StatusBar** control's **Panel** objects. The standard **Click** event also occurs when a **Panel** object is clicked.

The PanelClick event is only generated when the click occurs over a **Panel** object. When the **StatusBar** control's **Style** property is set to Simple style, panels are hidden, and therefore the PanelClick event is not generated.

Example

```
Global Enum sbrNoAutoSize = 0, sbrSpring,
  sbrContents
```

```
Ocx StatusBar sb
sb.Panels.Add , , "Hello" : sb.Panel(1).AutoSize =
  sbrNoAutoSize
sb.Add , , "Hello" : sb.Panel(2).AutoSize =
  sbrSpring
sb.Panels.Add , , "Hello" : sb.Panel(3).AutoSize =
  sbrContents
sb.Add , , "Hello" : sb(4).MinWidth = 50 :
  sb(4).AutoSize = sbrContents
Do : Sleep  : Until Me Is Nothing

Sub sb_PanelClick(Panel As Panel)
  Message "Single Click on:" & sb_Report(Panel)
EndSub

Sub sb_PanelDblClick(Panel As Panel)
  Message "Double Click on:" & sb_Report(Panel)
EndSub

Function sb_Report(Panel As Panel)
  Local a$ = " " & Panel.Text & #13#10 & "AutoSize
    state: "
  Select Panel.AutoSize
  Case sbrNoAutoSize : a$ = a$ & "sbrNoAutoSize(0)"
  Case sbrSpring : a$ = a$ & "sbrSpring(1)"
  Case sbrContents  : a$ = a$ & "sbrContents(2)"
  EndSelect
  Return a$
EndFunction
```

See Also

StatusBar

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Columns Property

Purpose

Returns or sets a value that determines whether a **ListBox** control scrolls vertically or horizontally and how the items in the columns are displayed. If it scrolls horizontally, the **Columns** property determines how many columns are displayed.

Syntax

object.**Columns** [= *number*]

Description

The settings for *number* are:

0(Default) Items are arranged in a single column and the ListBox scrolls vertically.

1 to n Items are arranged in snaking columns, filling the first column, then the second column, and so on. The **ListBox** scrolls horizontally and displays the specified number of columns.

Example

```
Local Int32 m, n
For n = 0 To 4
  Ocx Label lbl(n) = "Columns =" & n, 10, (n * 70)
    + 10, 60, 60
  Ocx ListBox lb(n) = n & " columns", 70, (n * 70)
    + 10, 200, 60 : lb(n).Columns = n
```

```
 For m = 1 To 20 : lb(n).AddItem "Item " &
   Format(m, "00") : Next m
Next n
Do : Sleep : Until Me Is Nothing
```

Remarks

See Also

ListBox

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

IntegralHeight Property

Purpose

Returns or sets a value indicating if the control displays partial items in a **ListBox** or **ComboBox**.

Syntax

*object***.IntegralHeight** [**=** *Boolean*]

object:ComboBox, ListBox Ocx

Description

The **IntegralHeight** property is False by default, the list doesn't resize itself even if the item is too tall to display completely. When set to True the list resizes itself to display only complete items.

If the number of items in a list exceeds what can be displayed, a scroll bar is automatically added to the control. You can prevent partial rows from being displayed by setting the **IntegralHeight** property to True.

Example

```
Local Int32 n
OpenW 1 : AutoRedraw = 1 : FontSize = 10
Debug FontSize
Text 10, 10, "IntegralHeight = True"
Ocx ListBox lb1 = "", 10, 30, 150, 200
Text 170, 10, "IntegralHeight = False"
Ocx ListBox lb2 = "", 170, 30, 150, 200
```

```
lb1.IntegralHeight = True
For n = 1 To 15 : lb1.AddItem "Item " & Format(n,
 "00") : lb2.AddItem "Item " & Format(n, "00") :
 Next n
Do : Sleep : Until Me Is Nothing
```

See Also

ListBox, ComboBox

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

ItemData Property

Purpose

Returns or sets a specific number for each item in a **ComboBox** or **ListBox** control.

Syntax

object.**ItemData(***index***)** [= *number*]

number:iexp

Description

The **ItemData** property is an array of long integer values with the same number of items as a control's **List** property. You can use the numbers associated with each item to identify the items. For example, you can use an employee's identification number to identify each employee name in a **ListBox** control. When you fill the **ListBox**, also fill the corresponding elements in the **ItemData** array with the employee numbers.

The **ItemData** property is often used as an index for an array of data structures associated with items in a **ListBox** control.

Example

```
Global a$, n As Int32
Ocx ListBox lb = "", 10, 10, 100, 200
For n = 1 To 26 : a$ = "Letter " & (n < 10 ? " " :
  "") & Trim(n) : lb.AddItem a$, n + 64 : Next n
```

```
Do : Sleep : Until Me Is Nothing

Sub lb_Click
  If lb.ListIndex <> -1
    Message lb.List(lb.ListIndex) & " is " &
      Chr(lb.ItemData(lb.ListIndex))
  EndIf
EndSub
```

Remarks

When you insert an item into a list with the **AddItem** method, an item is automatically inserted in the **ItemData** array as well. However, the value isn't reinitialized to zero; it retains the value that was in that position before you added the item to the list. When you use the **ItemData** property, be sure to set its value when adding new items to a list.

See Also

ListBox, ComboBox, Item, AddItem

{Created by Sjouke Hamstra; Last updated: 11/10/2014 by James Gaite}

List, ListCount, ListIndex Properties

Purpose

List returns or sets the items contained in a **ComboBox** or **ListBox** object's list portion. **ListCount** returns the number of items in the list portion of a control. **ListIndex** returns or sets the index of the currently selected item in the control.

Syntax

*object***.List***(index) [= string]*

*object***.ListIndex** *[= index%]*

*object***.ListCount**

object:ComboBox or ListBox Ocx

Description

The list is a string array in which each element is a list item. **List**(0) returns the first entry. The **List** property works in conjunction with the **ListCount** and **ListIndex** properties. Enumerating a list from 0 to **ListCount** -1 returns all items in the list.

ListIndex returns or sets the currently selected list item, it takes an index from 0 to **ListCount** - 1. When no item is selected, **ListIndex** returns -1.

Example

```
Debug.Show
~SetWindowPos(Debug.hWnd, 0, 205, 10, 300, 500, 0)
OpenW 1, 10, 10, 185, 350
Local n As Int32
Ocx ListBox lb = "", 10, 10, 150, 300 : .Sorted =
  False : .IntegralHeight = True
For n = 1 To 30 : lb.AddItem "List Item" & n :
  Next n
//Enumerate the list in Debug window
For n = 1 To lb.ListCount : Debug "List(" & Trim(n
  - 1) & ") = " & lb.List(n - 1) : Next n
Debug.Print
Do : Sleep  : Until Me Is Nothing
Debug.Hide

Sub lb_Click
  If lb.ListIndex <> -1 // An item has been
    selected
    Debug "Selected Item: "; lb.ListIndex; " -
      ";#34;lb.List(lb.ListIndex);#34
  EndIf
EndSub
```

Remarks

To specify items you want to display in a **ComboBox** or **ListBox** control, use the **AddItem** method. To remove items, use the **RemoveItem** method. To keep items in alphabetic order, set the control's **Sorted** property to **True** before adding items to the list.

See Also

ListBox, ComboBox

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

MultiSelect, Selected Properties (ListBox, ListView)

Purpose

MultiSelect returns or sets a Boolean value indicating whether a user can make multiple selections in a **ListBox.** The **Selected** property returns or sets the selection status of an item in a **ListBox** control.

Syntax

*object***.MultiSelect** [**=** *boolean*]

*object***.Selected(***index***)** [**=** *boolean*]

object:ListBox, ListView

Description

The **Selected**() property is an array of Boolean values with the same number of items as the **List** property.

Example

```
Form frm = "Listbox", , , 500, 400
Ocx ListBox lb1 =  "", 0, 0, 250, 200
.MultiSelect = 1
.TabStop = True
Ocx ListBox lb2 = "", 250, 0, 250, 200
.TabStop = True
Ocx Command cmd1 = "Add to 2", 100, 220, 80, 24
```

```
cmd1.Enabled = False
Dim i%
For i = 0 To Screen.FontCount - 1
  lb1.AddItem Screen.Fonts(i)
Next i
lb1.SetFocus
Do
  Sleep
Until Me Is Nothing

Sub cmd1_Click ()
  Dim i%
  lb2.Clear        ' Clear all items from the list.
  For i = 0 To lb1.ListCount - 1
    If lb1.Selected(i) Then
      lb2.AddItem lb1.List(i)
    End If
  Next i
End Sub

Sub lb1_Click
  ' The missing ListBox property: SelCount:
  Dim SelCount% = SendMessage(lb1.hWnd,
    LB_GETSELCOUNT, 0, 0)
  If SelCount = 0 && cmd1.Enabled
    cmd1.Enabled = False
  Else If SelCount > 0 && cmd1.Enabled = False
    cmd1.Enabled = True
  EndIf
EndSub
```

See Also

ListBox, ListView

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Find, FindExact, FindNext Method

Purpose

Searches a string in a **ListBox** or **ComboBox** Ocx control.

Syntax

i = object.**Find**(search$)

i = object.**FindNext**(start, search$)

i = object.**FindExact**(search$)

I, start:iexp
object:ListBox or ComboBox

Description

Find searches for a string and **FindNext** the next match of that string, as part of a list item in a **ListBox** or **ComboBox** Ocx control. With **FindNext**, if the string is not found in the remainder of the list, it will start again from the beginning of the list until a match is made or the *start* item is reached.

FindExact searches for an exact (but non-case-sensitive) match of a string with the whole of a list item in a **ListBox** or **ComboBox** Ocx control.

The return value is the index of the item.

Example

```
Dim list As Variant, n As Int32
list = Array("Matthew", "Mark", "Luke", "John",
  "Paul")
Ocx ListBox lbx = "", 10, 10, 150, 400 :
  lbx.Sorted = False
For n = 1 To 40 : lbx.AddItem list(Random(5)) &
  Iif(Random(2) = 0, " 2", "") : Next n
Ocx Label lbl = "String to Find:", 220, 10, 150,
  15 : lbl.BackColor = RGB(255, 255, 255)
Ocx ComboBox cmb = "", 220, 30, 150, 22 :
  cmb.Style = 2
For n = 0 To 4 : cmb.AddItem list(n) : Next n :
  cmb.ListIndex = 0
Ocx Command cmd1 = "Find", 190, 60, 60, 22
Ocx Command cmd2 = "Find Next", 260, 60, 60, 22
Ocx Command cmd3 = "Find Exact", 330, 60, 60, 22
Do : Sleep  : Until Me Is Nothing

Sub cmd1_Click
  // lbx.ListIndex =
    lbx.Find(cmb.List(cmb.ListIndex)) does not seem
    to work
  Local a$ = cmb.List(cmb.ListIndex)
  lbx.ListIndex = lbx.Find(a$)
EndSub

Sub cmd2_Click
  Local a$ = cmb.List(cmb.ListIndex)
  lbx.ListIndex = lbx.FindNext(lbx.ListIndex, a$)
EndSub

Sub cmd3_Click
  Local a$ = cmb.List(cmb.ListIndex)
  lbx.ListIndex = lbx.FindExact(a$)
EndSub
```

Remarks

See Also

ListBox, ComboBox

{Created by Sjouke Hamstra; Last updated: 17/11/2017 by James Gaite}

Style Property (ComboBox)

Purpose

Returns or sets a value indicating the display type and behavior of the control.

Syntax

Object.**Style** [= *value%*]

Object:ComboBox

Description

The **Style** property settings for the **ComboBox** control are:

Value	Description
0	(Default) Dropdown Combo. Includes a drop-down list and a text box. The user can select from the list or type in the text box.
1	Simple Combo. Includes a text box and a list, which doesn't drop down. The user can select from the list or type in the text box. The size of a Simple combo box includes both the edit and list portions. By default, a Simple combo box is sized so that none of the list is displayed. Increase the Height property to display more of the list.
2	Dropdown List. This style allows selection only from the drop-down list.

Example

```
Local a$, n As Int32
Ocx ComboBox cb1 = "", 10, 10, 150, 22 : .Style =
  0
Ocx ComboBox cb2 = "", 200, 10, 150, 150 : .Style
  = 1 : cb2.Height = 10 * TextHeight("A")
Ocx ComboBox cb3 = "", 390, 10, 150, 22 : .Style =
  2
For n = 1 To 20 : a$ = "Item " & Format(n, "00")
  cb1.AddItem a$
  cb2.AddItem a$
  cb3.AddItem a$
Next n
Do : Sleep : Until Me Is Nothing
```

Remarks

Use setting 0 (Dropdown Combo) or setting 1 (Simple Combo) to give the user a list of choices. Either style enables the user to enter a choice in the text box. Setting 0 saves space on the form because the list portion closes when the user selects an item.

Use setting 2 (Dropdown List) to display a fixed list of choices from which the user can select one. The list portion closes when the user selects an item.

See Also

ComboBox

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Scroll Event (ComboBox)

Purpose

Occurs after a scrollbar event.

Syntax

Sub *ComboBox*_**Scroll**()

Description

For a **ComboBox** control, this event occurs only when the scrollbars in the dropdown portion of the control are manipulated.

Example

```
Ocx ComboBox cmb = "", 10, 10, 100, 22 :
  cmb.Sorted = False
Local Int32 n
For n = 1 To 60 : cmb.AddItem "Item" & n : Next n
Do : Sleep : Until Me Is Nothing

Sub cmb_Scroll
  Message "Scrollbar moved"
EndSub
```

Known Issues

This event does not seem to work, as shown by the above example.

See Also

ComboBox

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Close Method, AutoClose Property , Close Event, Destroy Event

Purpose

The **Close** method closes a **Form** window. The **Close** event occurs when a window is about to close. The **AutoClose** property prevents automatic closure of a Form.

Syntax

Form.**Close**

Form.**AutoClose** [= value]

Sub *Form*_**Close**([index%,] Cancel?)

Sub *Form*_**Destroy**([index%])

value:Bool exp
index%:iexp, form number
Cancel?:boolean ByRef

Description

When **AutoClose** = 0 the form is not automatically closed when Alt-F4 is pressed, or when the close button in the caption is clicked. Instead, the program must handle the Form_**Close**(*Cancel*?) sub event to close the window by setting Cancel? = False. The ByRef parameter *Cancel*? is True by default, so that without changing it, the window isn't closed.

To explicitly close a window the **Close** method is available. This method will not result in invoking the **Close** event sub. A window can also be closed by setting its object variable to Nothing (Set Win_1 = Nothing).

After closing a window/form/dialog the **Destroy** event is invoked. This is the place to release resources and finalize the Form. The **Destroy** event is generated when *DestroyWindow* is called. Windows doesn't send WM_CLOSE and WM_DESTROY messages when the user logs off. The **QueryEndSession** event is the time to do the final things.

Example

```
OpenW 1
Me.AutoClose = 0
Do
  Sleep
Until Me Is Nothing

Sub Win_1_Close(Cancel?)
  If MsgBox("Close Form?", MB_YESNO) = IDYES Then C
    ancel? = False
EndSub

Sub Win_1_Destroy
  ' release resources
EndSub
```

Remarks

For all forms **AutoClose** = True by default, except for Dialogs, where **Dlg_n.AutoClose** = False.

In addition, unlike **CloseW**, if Form.**Close** is used for a form which does not exist or has been set to Nothing, an error is returned.

See Also

Form, QueryEndSession

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

ControlBox Property

Purpose

Returns or sets a value indicating whether a Control-menu box is displayed on a form.

Syntax

object.**ControlBox** [= True | False]

Description

To display a Control-menu box, you must also set the form's **BorderStyle** property to 1 (**basFixed**), or 2 (**basThick**).

When **ControlBox** = 0 the **MinButton**, **MaxButton**, and **HelpButton** are removed as well. These properties depend on each other. When **Caption** = "" as well, no title bar is drawn.

Example

```
OpenW 1
Print "Press any key to remove ControlBox"
Local a% : KeyGet a%
Win_1.ControlBox = False
Cls
Print "Press any key to close"
KeyGet a%
CloseW 1
```

Remarks

Although **ControlBox** = False disables the system menu, a **Form** can still be moved when it has caption by clicking and holding the mouse button in the title bar. Also, by double clicking the title bar the form is maximized or minimized, respectively.

See Also

Form, MinButton, MaxButton, HelpButton, BorderStyle

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

Controls Property, Control Ocx

Purpose

Returns a reference to a collection of **Control** objects on a **Form**.

Syntax

Form.**Controls**

Control

Description

A collection of type **Control**. The collection can be iterated over using **For Each**. Furthermore, it provides the **Count** property.

The **Control** type as a generic variable type for controls. When you declare a variable **As Control**, you can assign it a reference to any control. You cannot create an instance of the **Control** class.

Example

```
Form frm1 = , 0, 0, 150, 200
// Populate Form
Ocx Command cmd = "Command", 10, 10, 100, 22
Ocx Option opt(1) = "Option 1", 10, 40, 100, 14
Ocx Option opt(2) = "Option 2", 10, 60, 100, 14
Ocx CheckBox checkbox = "Checkbox", 10, 85, 100,
 14
// Display Control properties in Debug screen
Debug.Show
~SetWindowPos(Debug.hWnd, 0, 250, 0, 500, 500, 0)
Dim o As Control
Trace frm1.Controls.Count
Debug.Print
For Each o In frm1.Controls
  Trace o.Name
  Try
    Trace o.Index
  Catch
  EndCatch
  Debug
Next
Do : Sleep : Until frm1 Is Nothing
Debug.Hide
```

Remarks

Accessing properties and methods of a control is faster if you use a variable declared with the same type as the control (for example, **As TreeView** or **As Command**), because GFA-BASIC 32 can use early binding. GFA-BASIC 32 must use late binding to access properties and methods of a control assigned to a variable declared **As Control**.

See Also

Form, Forms

{Created by Sjouke Hamstra; Last updated: 26/09/2014 by James Gaite}

hDC2 Property

Purpose

Returns a handle provided by the Microsoft Windows operating environment to memory device context of the **AutoRedraw** image of a **Form**.

Syntax

Form.**hDC2**

Form:Form Object

Description

This property is a Windows operating environment device context handle. The Windows operating environment manages the system display by assigning a device context for each form in your application. The **AutoRedraw** property requires another DC; a memory device context to draw the graphic output in a memory bitmap. You can use the **hDC2** property to refer to the handle for the Form's memory device context. This provides a value to pass to Windows API calls.

The memory graphic image can be obtained with the **Image** property of the **Form**.

Example

```
OpenW 1, 0, 0, 400, 400
AutoRedraw = 1
Print "This is window 1"
```

```
OpenW 2, 401, 0, 400, 400
BitBlt Win_1.hDC2, 0, 0, 400, 400, Win_2.hDC, 0,
  0, &H00CC0020
```

Remarks

The value of the **hDC2** property can change while a program is running, so don't store the value in a variable; instead, use the **hDC2** property each time you need it.

See Also

Form, hDC, AutoRedraw, Image, _DC(), _DC2

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

MaxButton, MinButton, HelpButton Property

Purpose

Sets the minimize, maximize and help button of a Form.

Syntax

Form.**MaxButton** [= True | False]

Form.**MaxButton** [= True | False]

Form.**HelpButton** [= True | False]

Description

When set to True the form has a Maximize or Minimize button (default).

A Maximize button enables users to enlarge a form window to full-screen size. To display a Maximize button, you must also set the form's **BorderStyle** property to either 1 (Fixed Single), 2 (Sizable), or 3 (Fixed Double).

The settings you specify for the **MaxButton**, **MinButton**, **BorderStyle**, and **ControlBox** properties are related with each other. When **ControlBox** = False the **MinButton** and **MaxButton** aren't visible.

HelpButton removes or sets the help button in the title bar of a **Form**. To get the What's This question mark button in the title bar of the window, the properties of both **MinButton** and **MaxButton** must be set to False. The

OnCtrlHelp event occurs when the help button cursor is clicked on an Ocx control or F1 is pressed. Set the WhatsThisHelpID of a control to identify the help content.

Instead of using **HelpButton**, it is possible to create the same effectby using a pushbutton/command control which can toggle the window's WhatsThisMode.

Example

```
OpenW 1
Me.MinButton = 0
Me.MaxButton = 0
Me.HelpButton = True
Ocx Command cmd = "Show Max && Min buttons", 10,
  40, 140, 24
cmd.WhatsThisHelpID = 1001
Do
  Sleep
Until Me Is Nothing

Sub cmd_Click
  Me.MaxButton = Not Me.MaxButton
  Me.MinButton = Not Me.MinButton
  cmd.Caption = (Me.MaxButton ? "Hide" : "Show") &
    " Max && Min buttons"
EndSub

Sub Win_1_OnCtrlHelp(Ctrl As Object, x%, y%)
  Trace Ctrl.WhatsThisHelpID
EndSub
```

Remarks

See Also

Form

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

hMdiClientWnd Property

Purpose

Returns the handle of the MDICLIENT window of the MDI parent window.

Syntax

h = *Form*.**hMdiClientWnd**

h:Handle

Description

Only valid when the window is a MDI parent window.

Example

```
ParentW 1
Dim hMdiClient As Handle
hMdiClient = Win_1.hMdiClientWnd
Debug.Show
Trace Hex(hMdiClient)
Do
  Sleep
Until Me Is Nothing
```

See Also

Form, hWnd, MdiParent, MdiChild, ChildW, ParentW

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

HScMax, HScMin, HScPos, HScPage, HScStep, HScTrack Properties

Purpose

Sets and returns the horizontal scrollbar values for a Form.

Syntax

Form.**HScMax** [= value]

Form.**HScMin** [= value]

Form.**HScPos** [= value]

Form.**HScPage** [= value]

Form.**HScStep** [= value]

Form.**HScTrack** [= value]

value : Long exp

Description

Properties used to the horizontal scrollbar of a Form object.

HScPos is the value of the control, and can range from **HScMin** to **HScMax**, inclusive. It designates where the scroll bar button is positioned along the scroll bar.

HScMin is a number specifying the minimum value that the scroll bar can have. This number ranges from 0 to 30,000, but cannot be greater than the maximum value given in **HScMax**.

HScMax is a number specifying the maximum value that the scroll bar can have. This number ranges from 0 to 30,000. Setting **HScMax** to 0 makes the scroll bar disappear. To disable the scroll bar but keep it visible use `~EnableScrollBar(hWnd, SB_HORZ, ESB_DISABLE_BOTH)` and to enable it again use `~EnableScrollBar(hWnd, SB_HORZ, ESB_ENABLE_BOTH).`

HScStep is a number specifying the increment that the value is adjusted by when the scrollbar arrow is clicked.

HScPage is a number specifying the increment that the value is adjusted by when the page scroll region of a scroll bar is clicked.

HScTrack returns the current position of the scrollbar in the **_HScrolling** event sub. This sub called only when the thumb is being moved. The _HScroll event sub is called after the scrolling is complete.

Note: If you manually change either **HScPos** or **HScTrack**, you MUST adjust the value of the other; if not, the scrollbars will display odd and incorrect behaviour. Also, using the **SetFocus** method will reset both of these values to zero.

Default values: .**HScPos** = 0, .**HScTrack** = 0, .**HScMin** = 0, .**HScStep** = 1, **HScPage** = 100, .**HScMax** = 1000.

Example

```
Global Int32 a = 160, b = a / 2, n
Global Int32 iw = (5 * a) + 100 // Width of the
  actual work area
Global Int32 vw = (2 * a) + b  // Width of visible
  area within window
OpenW Fixed 1, , , vw + (Screen.cxFixedFrame * 2),
  500 : Win_1.ControlBox = False
For n = 0 To 4 : Ocx Command cmd(n) = "Close
  Button " & n, ((n * a) + b), 200, 100, 22 : Next
  n
Me.ScrollBars = basHorizontal
Me.HScMin = 0
Me.HScStep = b / 2
Me.HScPage = a / 2
Me.HScMax = iw - vw + Me.HScPage // Width of Work
  Area - Width of Visible Area + HScPage
Do
  Sleep
Until Me Is Nothing

Sub Win_1_HScroll
  For n = 0 To 4 : cmd(n).Left = (((n * 200) + 100)
    - Me.HScPos) : Next n
EndSub

Sub Win_1_HScrolling
  For n = 0 To 4 : cmd(n).Left = (((n * 200) + 100)
    - Me.HScTrack) : Next n
EndSub

Sub cmd_Click(Index%)
  Win_1.Close
EndSub
```

Remarks

See Also

Form, _HScrolling, _HScroll, VScMax, VScMin, VScPos, VScPage, VScStep, VScTrack, _VScrolling, _VScroll

{Created by Sjouke Hamstra; Last updated: 08/03/2018 by James Gaite}

IsDialog Property

Purpose

Returns True when a Form is created using the **Dialog** command.

Syntax

? = Form.**IsDialog**

Description

Returns true when a Form is created using the **Dialog** command. This is useful because of different management of dialogs as forms.

For a dialog-form **AutoClose** = 0 and the background color is white.

Example

```
Dialog # 1, 50, 50, 200, 110, "DlgBase Outside",
  $80 ', -12, "ARIAL"
  LText "This is should be bold!", 3, 32, 16, 350,
    16, $0
  PushButton "Close", IDOK,  55, 45, 80, 20
EndDialog
ShowDialog # 1
Trace Dlg_1.IsDialog   // returns False
```

Remarks

This property doesn't seem to work.

See Also

Dialog, Form

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

MdiChild, MdiParent Properties

Purpose

For MDI windows return True when a Form is a **MdiChild** or a **MdiParent** Form.

Syntax

? = *Form*.**MdiChild**

? = *Form*.**MdiParent**

Description

A Form can only have one of these properties set. These properties can only be set in the Form Editor. At run time they can be inquired only.

Example

```
ParentW 1
ChildW 2, 1
Print Me.MdiChild
Print Me.MdiParent
```

Remarks

The **Owned** and **MdiChild** properties cannot be combined.

See Also

ChildW, ParentW, Form

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

Moveable, Sizeable Properties

Purpose

Returns or sets a value which specifies if the Form can be moved or be resized.

Syntax

Form.**Moveable = ***boolean*

Form.**Sizeable = ***boolean*

Description

When **Moveable** = False the Form cannot be moved. Be aware, when there are a Caption, a ControlBox, or a MinButton or MaxButton a window should be moveable.

When **Sizeable** is True the **BorderStyle** is changed to 2, and when **Sizeable** is set to False the **BorderStyle** is set to 1.

Example

```
OpenW 1, 10, 10, 300, 300
Ocx CheckBox chk(1) = "Fix Window Position", 10,
  10, 140, 22
Ocx CheckBox chk(2) = "Window Resizable", 10, 40,
  140, 22 : chk(2).Value = 1
Do
  Sleep
```

```
Until Win_1 Is Nothing

Sub chk_Click(Index%)
  Win_1.Moveable = Not (-chk(1).Value)
  Win_1.Sizeable = - chk(2).Value
EndSub
```

See Also

Form, Sizeable, Caption, ControlBox, MinButton, MaxButton

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

ToTop, ToBack Methods, OnTop Property

Purpose

ToTop places the Form at the top of the Z order. **ToBack** places the Form at the bottom of the Z order.

Syntax

Form.**ToBack**

Form.**ToTop**

Form**.OnTop** [= Boolean]

Description

The **ToTop** and **ToBack** methods use the *SetWindowPos* function to change the Z order of Form. Child, pop-up, and top-level windows are ordered according to their appearance on the screen. The topmost window receives the highest rank and is the first window in the Z order. If a Form identifies a topmost window, with **ToBack** the window loses its topmost status and is placed at the bottom of all other windows.

The **OnTop** property sets the topmost state of a window. **OnTop** places the window above all non-topmost windows. The window maintains its topmost position even when it is deactivated; using the **ToTop** method does NOT set the corresponding **OnTop** property to True.

Example

```
OpenW 1 : Win_1.Caption = "Window 1"
Ocx Command cmd(1) = "Send to Back", 10, 10, 120,
  22
Ocx Label lbl(1) = "", 10, 40, 120, 14
OpenW 2 : Win_2.Caption = "Window 2"
Ocx Command cmd(2) = "Send to Back", 10, 10, 120,
  22
Ocx Label lbl(2) = "", 10, 40, 120, 14
Do : Sleep  : Until Win_1 Is Nothing Or Win_2 Is
  Nothing
CloseW 1 : CloseW 2

Sub cmd_Click(Index%)
  If Index% = 1 Then Win_2.ToTop // or Win_2.OnTop
    = True
  If Index% = 2 Then Win_2.ToBack (* or Win_1.OnTop
    = True *) : Win_1.ToTop
EndSub

Sub Win_1_Activate
  cmd(2).Caption = ""
  cmd(1).Caption = "Send to Back"
  lbl(1).Caption = "Window 1 on Top"
EndSub

Sub Win_2_Activate
  cmd(1).Caption = ""
  cmd(2).Caption = "Send to Back"
  lbl(2).Caption = "Window 2 on Top"
EndSub
```

Remarks

The **ZOrder** method changes the z-order as well.

See Also

[Form]

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Paint Event; PaintLeft, PaintTop, PaintWidth, and PaintHeight Properties

Purpose

The **Paint** event occurs when a **Form** is first displayed or when part or all of a Form is exposed after being moved or enlarged, or after a window that was covering the object has been moved.

Syntax

Sub Form_**Paint**([index%])

x! = Form.**PaintLeft**
x! = Form.**PaintTop**
x! = Form.**PaintWidth**
x! = Form.**PaintHeight**

index%:iexp, Form number
x!Single exp

Description

A **Paint** event procedure is useful if you have output from graphics methods in your code. With a **Paint** procedure, you can ensure that such output is repainted when necessary. The **Paint** event is also invoked when the **AutoRedraw** property is set to **True**.

The **Paint** event is invoked when part or all of the client area has been invalidated and/or the **Refresh** method is used.

The **PaintLeft**, **PaintTop**, **PaintWidth**, and **PaintHeight** properties specify the invalidated area, the area that needs updated. These properties are only valid inside the **Paint** event.

Example

```
OpenW 1, 10, 10, 300, 300
Ocx CheckBox chk = "Invalidate area 10,10 to
 110,110", 10, 10, 180, 14
Do
  Sleep
Until Win_1 Is Nothing

Sub chk_Click
  If chk.Value = 1
    Win_1.Invalidate 10, 10, 100, 100
  Else
    Win_1.InvalidateAll
  EndIf
EndSub

Sub Win_1_Paint
  Print AT(1, 4); "PaintLeft "; Me.PaintLeft;
    Space(10)
  Print AT(1, 5); "PaintTop "; Me.PaintTop;
    Space(10)
  Print AT(1, 6); "PaintWidth "; Me.PaintWidth;
    Space(10)
  Print AT(1, 7); "PaintHeight "; Me.PaintHeight;
    Space(10)
EndSub

Sub Win_1_ReSize
  // Resizing the Window invokes InvalidateAll
  chk.Value = 0
EndSub
```

Resize the window and watch the changes in the Paint area dimensions.

Note that when you Invalidate the area 10, 10 to 110, 110 then **PaintTop** actually reads 24 and **PaintHeight** 86; this is due to the position of the checkbox area which is not included in the Invalidate statement.

Remarks

A **ClearW** command is not allowed in a **Paint** event, because it generates a WM_PAINT message.

See Also

Form

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

ShowInTaskbar, StartupMode Properties

Purpose

ShowInTaskbar returns or sets a value that determines whether a Form object appears in the Windows taskbar. Read-only at run time.

StartupMode returns or sets a value specifying the position of a Form object when it first appears with **LoadForm**. Not available at run time.

Syntax

Form.**ShowInTaskbar**

% = *Form*. **StartupMode**

Description

Use the **ShowInTaskbar** property to keep dialog boxes in your application from appearing in the taskbar. Only available at design time in the Form Editor. (It is a hidden property in the code editor.)

Use the **StartupMode** property to specify the position of the Form object at design time in the Form Editor. The **StartupMode** property can take the following values:

0 - no initial setting

1 - centered on the screen

2 - maximized

The optional parameters of the **LoadForm** command can overrule the **StartupMode** setting.

Example

```
// Design a form in the Form Editor and title it
  frm1
// Then run the following code.
LoadForm frm1
AutoRedraw = 1
Print Me.StartupMode
Do : Sleep : Until frm1 Is Nothing
```

Remarks

Methods of recreating the **StartupMode** settings for forms created 'in-program' are:

- *frm1*.**StartupMode** = 1 (Centred) => **Form Center** *frm1*
- *frm1*.**StartupMode** = 2 (Maximized) => **Form Full** *frm1*

See Also

LoadForm, Form

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

VScMax, VScMin, VScPos, VScPage, VScStep, VScTrack Properties

Purpose

Sets and returns the vertical scrollbar values for a Form.

Syntax

Form.**VScMax** [= value]

Form.**VScMin** [= value]

Form.**VScPos** [= value]

Form.**VScPage** [= value]

Form.**VScStep** [= value]

Form.**VScTrack** [= value]

value : Long exp

Description

Properties used to set the vertical scrollbar of a Form object.

VScPos is the value of the control, and can range from **VScMin** to **VScMax**, inclusive. It designates where the scroll bar button is positioned along the scroll bar.

VScMin is a number specifying the minimum value that the scroll bar can have. This number ranges from 0 to 30,000, but cannot be greater than the maximum value given in **VScMax**.

VScMax is a number specifying the maximum value that the scroll bar can have. This number ranges from 0 to 30,000. Setting **VScMax** to 0 makes the scroll bar disappear. To disable the scroll bar but keep it visible use `~EnableScrollBar(hWnd, SB_VERT, ESB_DISABLE_BOTH)` and to enable it again use `~EnableScrollBar(hWnd, SB_VERT, ESB_ENABLE_BOTH)`.

VScStep is a number specifying the increment that the value is adjusted by when the scrollbar arrow is clicked.

VScPage is a number specifying the increment that the value is adjusted by when the page scroll region of a scroll bar is clicked.

VScTrack returns the current position of the scrollbar in the **_VScrolling** event sub. This sub called only when the thumb is being moved. The _VScroll event sub is called after the scrolling is complete.

Note: If you manually change either **VScPos** or **VScTrack**, you MUST adjust the value of the other; if not, the scrollbars will display odd and incorrect behaviour. Also, using the **SetFocus** method will reset both of these values to zero.

Default values: .**VScPos** = 0, .**VScTrack** = 0, .**VScMin** = 0, .**VScStep** = 1, **VScPage** = 100, .**VScMax** = 1000.

Example

```
Global Int32 a = 160, b = a / 2, n
Global Int32 ih = (5 * a) + 22 // Height of the
  actual work area
Global Int32 vh = (2 * a) + b  // Height of
  visible area within window
OpenW Fixed 1, , , 500, vh + (Screen.cyFixedFrame
  * 2) + Screen.cyCaption : Win_1.ControlBox =
  False
For n = 0 To 4 : Ocx Command cmd(n) = "Close
  Button " & n, 200, ((n * a) + b), 100, 22 : Next
  n
Me.ScrollBars = basVertical
Me.VScMin = 0
Me.VScStep = b / 2
Me.VScPage = a / 2
Me.VScMax = ih - vh + Me.VScPage // Height of Work
  Area - Height of Visible Area + VScPage
Do
  Sleep
Until Me Is Nothing

Sub Win_1_VScroll
  For n = 0 To 4 : cmd(n).Top = (((n * a) + b) -
    Me.VScPos) : Next n
EndSub

Sub Win_1_VScrolling
  For n = 0 To 4 : cmd(n).Top = (((n * a) + b) -
    Me.VScTrack) : Next n
EndSub

Sub cmd_Click(Index%)
  Win_1.Close
EndSub
```

Remarks

See Also

Form, _HScrolling, _HScroll, HScMax, HScMin, HScPos, HScPage, HScStep, HScTrack, _VScrolling, _VScroll

{Created by Sjouke Hamstra; Last updated: 08/03/2018 by James Gaite}

WindowState Property

Purpose

Returns or sets a value indicating the visual state of a form window at run time.

Syntax

Form.**WindowState** [**=** *value*]

value:iexp

Description

value is a constant (an integer) specifying the state of the object. The Form can be minimized, maximized, or normal. The constants are:

basNormal = 0, when set is equal to Form.**Restore**

basMinimized = 1, when set is equal to Form.**Minimize**

basMaximized = 2, when set is equal to Form.**Maximize**

When **WindowState** is set, the state of the window is immediately updated.

Example

```
OpenW 1 : AutoRedraw = 1
Me.WindowState = basMaximized
Print Win_1.WindowState
```

See Also

Form, Iconic?, Zoomed?

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Activate, Deactivate Events

Purpose

Activate - occurs when an object becomes the active window.

Deactivate - occurs when an object is no longer the active window.

Syntax

Sub *Form*_**Activate(** [*Index*%] **)**

Sub *Form*_**Deactivate(**[*Index*%]**)**

Description

An object can become active by user action, such as clicking it, or by using the **Show** or **SetFocus** methods in code.

The Activate event can occur only when an object is visible

The Activate and Deactivate events occur only when moving the focus within an application. Moving the focus to or from an object in another application doesn't trigger either event. The Deactivate event doesn't occur when unloading an object.

The Activate event occurs before the GotFocus event; the LostFocus event occurs before the Deactivate event.

These events occur for MDI child forms only when the focus changes from one child form to another. In an **MdiParent** form object with two child forms, for example, the child forms receive these events when the focus moves between them. However, when the focus changes between a child form and a non-MDI child form, the parent MDI Form receives the Activate and Deactivate events.

Example

```
Form frm1 = "Activate, Deactivate Events", 20, 20,
  300, 300
Do
  Sleep
Until Me Is Nothing

Sub frm1_Activate
  Print "Form Activated"
EndSub

Sub frm1_Deactivate
  Print "Form Deactivated"
EndSub
```

See Also

Form, GotFocus, LostFocus, SetFocus, Activate

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Enable, Disable Methods

Purpose

Enables or disables mouse and keyboard input for a **Form**.

Syntax

[*Form*.]**Enable**

[*Form*.]**Disable**

Description

The mouse and keyboard input for windows can be controlled separately. **Enable** enables these inputs for the form, **Disable** disables them.

Example

```
OpenW # 1
Win_1.Disable
Print Me.Enabled   // prints 0 (False)
// ... Now no input (mouse & keyboard) possible
Me.Enable
Print Me.Enabled   // prints -1 (True)
```

Now input (mouse & keyboard) is possible again. The input for window 1 is first deactivated and the reactivated.

See Also

Form, Enabled

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

Hide, Show Methods

Purpose

Hides or shows an object.

Syntax

object.**Hide**

object.**Show**

object:Ocx object

Description

The **Hide** method hides an object at run time. **Show** makes it visible again. Invoking these methods in code enables you to hide and later redisplay an Ocx object at run time in response to a particular event.

At design time, to set the visibility at startup, set the **Visible** property to **False.**

Example

```
OpenW Center 100, , , 140, 100
Ocx Command cmd = "Hide this window", 10, 20, 100,
  22
Do  : Sleep : Until Form(100) Is Nothing

Sub cmd_Click
  If Left(cmd.Caption, 4) = "Hide"
    Form(100).Hide
    cmd.Caption = "Close this window"
    Message "Click OK to show the window again"
    Form(100).Show
  Else
    Form(100).Close
  EndIf
EndSub
```

Remarks

Using the **Show** or **Hide** method on a form is the same as setting the form's **Visible** property in code to **True** or **False**, respectively.

See Also

Form, Visible

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

Invalidate, InvalidateAll method

Purpose

Marks a specified rectangle of a **Form** for redraw.

Syntax

Form.**Invalidate** [x!, y!, w!, h!]

Form.**InvalidateAll**

x! ,y! ,w! ,h!:single expression

Description

The **Invalidate** method is used when a rectangular area of form is to be redrawn. The upper left corner of the rectangle is specified in x! and y!, the width in w!, and the height in h!. The coordinates are in reference to form's scale mode.

The system sends a WM_PAINT message to a window whenever its update region is not empty and there are no other messages in the application queue for that window. The message is processed in the next **Sleep** command, which then invokes the **Paint** event sub.

InvalidateAll invalidates the entire client area and is equal to **Invalidate** without parameters, but a bit faster, though.

Example

```
OpenW 1 , , , 400, 400
```

```
// Without AutoRedraw, the window does not store
  the printed rectangle
' Win_1.AutoRedraw = 1
Ocx Command cmd1 = "Invalidate Top && Left", 20,
  10, 120, 22
Ocx Command cmd2 = "Invalidate Everything", 20,
  50, 120, 22
Ocx Command cmd3 = "Redraw Rectangle", 20, 80,
  120, 22
Box 10, 40, 200, 200
Do : Sleep : Until Win_1 Is Nothing

Sub cmd1_Click
  // Repaints an area covering the top and left of
    the rectangle...
  // ...from an image which doesn't contain the
    rectangle
  Win_1.Invalidate 10, 40, 189, 159
EndSub

Sub cmd2_Click
  // Repaints the whole window from an image...
  // ...which doesn't contain the rectangle
  Win_1.InvalidateAll
EndSub

Sub cmd3_Click
  Box 10, 40, 200, 200
EndSub
```

Example 2

```
OpenW 1, 10, 10, 200, 200
OpenW 2, 220, 10, 200, 200
Print "Text to be copied"
Ocx Command cmd = "Copy Text", 10, 40, 100, 22
```

```
Do : Sleep : Until Win_1 Is Nothing Or Win_2 Is
 Nothing
CloseW 1
CloseW 2

Sub cmd_Click
  // AutoRedraw for Window 1 can be switched on
    here or when the window is opened
  Win_1.AutoRedraw = 1
  // Coies the contents of Window 2 to the hDC2 of
    Window 1
  BitBlt Win_2.hDC, 0, 0, 200, 200, Win_1.hDC2, 0,
    0, SRCCOPY
  // Forces Window 1 to repaint from the bitmap
    image copied to hDC2
  Win_1.Invalidate
EndSub
```

Remarks

Invalidate[All] sends a redraw message for a rectangular area. **Invalidate** corresponds to Windows function *InvalidateRect*().

See Also

Form, Validate, Scale, _Paint

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Maximize, Minimize, Restore Methods

Purpose

Causes the **Form** to be maximized or minimized.

Syntax

Form.**Maximize**

Form.**Minimize**

Form.**Restore**

Description

The **Maximize** and **FullW** methods maximize the form and are identical. **Minimize** minimizes the form.

Restore brings the window back to the size before Maximize, Minimize, or FullW brought it.

Example

```
OpenW 1
Message "Click OK to Maximize Window"
Win_1.Maximize
Message "Click OK to Restore Window"
Win_1.Restore
Message "Click OK to Minimize Window"
Win_1.Minimize
Message "Click OK to Restore Window again"
Win_1.Restore
```

```
Message "Click OK to Close Window"
CloseW 1
```

See Also

Form

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

MdiCascade, MdiTile, MdiIconArrange Methods

Purpose

Arrange the MDI child windows in the MDI parent form.

Syntax

Form.**MdiCascade** [flag%]

Form.**MdiTile** [flag%]

Form.**MdiIconArrange**

Form:Parent MDI Form

Description

To arrange child windows in the cascade format, use the **MdiCascade** method on the parent form. Typically, the application uses the method when the user clicks **Cascade** on the **Window** menu. The optional parameter flag specifies a cascade flag. The only flag currently available, MDITILE_SKIPDISABLED, prevents disabled MDI child windows from being cascaded.

To arrange child windows in the tile format, use the **MdiTile** method on the parent form. Typically, the application sends this message when the user clicks **Tile** on the **Window** menu. The optional parameter specifies a tiling flag. This parameter can be one of the following values:

ValueMeaning

MDITILE_VERTICAL (0) - Tiles MDI child windows so that they are tall rather than wide.

MDITILE_HORIZONTAL (1) - Tiles MDI child windows so that they are wide rather than tall.

MDITILE_SKIPDISABLED (2) - Prevents disabled MDI child windows from being tiled.

The system automatically displays a child window's icon in the lower portion of the client window when the child window is minimized. Use the **MdiIconArrange** method on the parent form. Typically, the application sends this message when the user clicks **Arrange Icons** on the **Window** menu.

Example

```
ParentW 1
Ocx StatusBar stb
Dim m$(), i%
Array m$() = "File"#10 "New"#10 "-"#10
  "Exit"#10#10 _
  "Edit"#10#10 "Window"#10 "#1000#Cascade"#10 _
  "#1001#&Tile Vertical"#10 "#1002#Tile
   Horizontal"#10 _
  "#1003#Next Window"#10 "#1004#&Previous
   Window"#10#10 _
  "Help"#10 "#1005#About"#10#10
Menu m()
Me.MdiSetMenu 2
For i = 2 To 7
  ChildW i, 1
  Me.Caption = "MDI Child #" & Format(i)
Next
Do
  Sleep
```

```
Loop Until Me Is Nothing

Sub Win_1_MenuOver(Idx%)
  stb.SimpleText = Idx < 0 ?  "" : Dec(Idx)
EndSub

Sub Win_1_MenuEvent(Idx%)
  Switch Idx
  Case 3
    If MsgBox("Quit Program?", MB_YESNO) = IDYES _
      Win_1.Close
  Case 1000 : Win_1.MdiCascade
  Case 1001 : Win_1.MdiTile
  Case 1002 : Win_1.MdiTile 1
  Case 1003 : Win_1.MdiNext
  Case 1004 : Win_1.MdiPrev
  Case 1005 : MsgBox "GFA-BASIC 32 MDI Demo"
  EndSwitch
End Sub
```

Remarks

An MDI application can arrange its child windows in either a cascade or tile format. When the child windows are cascaded, the windows appear in a stack. The window on the bottom of the stack occupies the upper left corner of the screen, and the remaining windows are offset vertically and horizontally so that the left border and title bar of each child window is visible.

When the child windows are tiled, the system displays each child window in its entirety - overlapping none of the windows. All of the windows are sized, as necessary, to fit within the client window. An MDI application should provide a different icon for each type of child window it supports.

The application specifies an icon when registering the child window class.

See Also

Form, ParentW, ChildW, OpenW, MdiSetMenu, MdiNext, MdiPrev, MdiActivate, MdiGetActive

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

MdiGetActive, MdiActivate, MdiNext, MdiPrev Methods

Purpose

MdiActivate, **MdiNext**, and **MdiPrev** activate a MDI child window. **MdiGetActive** returns the current active child window.

Syntax

ChildForm.**MdiActivate**

MDIForm.**MdiGetActive**

MDIForm.**MdiNext**

MDIForm.**MdiPrev**

ChildForm:MdiChild Form
MDIForm:MdiParent Form

Description

MdiActivate is a method to be used with a **MdiChild** form. It activates the child window and brings it to the front. **MdiGetActive** returns the current active child window.

MdiNext and **MdiPrev** activate the next (Ctrl+F6) or previous child window. These methods are to be performed on the parent window.

Example

See MdiCascade example.

Remarks

The menu entries for the child windows have identifier values starting from 64000.

See Also

MdiCascade, Form, ParentW, ChildW, OpenW

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

MdiSetMenu Method

Purpose

Inserts a special MDI parent window menu.

Syntax

Form.**MdiSetMenu** [n%]

Form:Parent window
n%:iexp

Description

The MDI window menu handles the list of open child windows. The parameter n% specifies the submenu to append the MDI window list. Normally, this is the Window submenu which offers options to arrange the child windows. **MdiSetMenu** without a parameter or when n < 0 will disable the automatic handling of the child windows.

Example

See MdiCascade example.

Remarks

The menu entries for the child windows have identifier values starting from 64000.

See Also

MdiCascade, Form, ParentW, ChildW, OpenW

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Owned Property, Owner Method

Purpose

The **Owned** property determines that a form is to be owned and is only available in the Form Editor.

Owner returns the **Form** object for the owner of a form.

Syntax

Set f = Form.**Owner**

Description

The **Owned** property can only be set in Form Editor. At design time you can set the **Owned** property determining that the form is to be loaded as an owned window. When set and when executing **LoadForm**, the form will be owned by the current active window (**Me**). When **Me** = Nothing at the time of execution of **LoadForm** the **Owned** property is ignored.

When you use this option, you achieve two interesting effects: the owned form is always shown in front of its owner (parent), even if the parent has the focus, and when the parent form is closed or minimized, all forms it owns are also automatically closed or minimized. You can take advantage of this feature to create floating forms that host a toolbar, a palette of tools, a group of icons, and so on. This technique is most effective if it is combined with the window state options **Fixed** and/or **Tool**/**Palette**. These option are specified with the **LoadForm** statement.

When a form is created with an owner, the owner object can be obtained with the **Owner** method.

Example

```
OpenW 1
OpenW Owner Me, 2
AutoRedraw = 1
Dim f As Form
Set f = Win_2.Owner
Trace f          ' Form(Win_1)
Do
  Sleep
Until Win_1 Is Nothing Or Win_2 Is Nothing
CloseW 1 : CloseW 2
Debug.Show
```

Remarks

Owned is a hidden property and not available in code.

A form can be created as being owned in code, when the **Owner** *frm* clause is used.

See Also

Form Object, Form, LoadForm, OpenW

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

PrintForm Method, PrintFormWidth, PrintFormHeight Properties

Purpose

PrintForm prints a **Form** object using the current printer. The **PrintFormWidth** and **PrintFormHeight** properties return the size of the image.

Syntax

Form.**PrintForm**[(f%)]

w! = *Form*.**PrintFormWidth**[(f%)]

h! = *Form*.**PrintFormHeight**[(f%)]

f%:iexp, 0 or 1
w!, h!:Single exp

Description

PrintForm prints all visible objects and bitmaps of the **Form** object. **PrintForm** also prints graphics added to a Form object at run time if the **AutoRedraw** property is True when the graphics are drawn.

The printer used by **PrintForm** is determined by the **Printer** object. The image is printed without taking the borders into account. A **StartDoc** is automatically executed when currently no print job is opened. The image is printed at Printer.**CurrentX** = 0 and Printer.**CurrentY**. The size of the image is proportional (formwidth/screenwidth = printwidth/printerwidth). The *printwidth* is calculated from the Printer.**Width** and Printer.**Height** properties. However, when the optional flag *f%* = 1, the width is calculated using Printer.**PageWidth** and Printer.**PageHeight**.

Multiple forms can be printed next to each other by setting Printer.**Left**.

Once the Printer is initialized, the **PrintFormWidth** and **PrintFormHeight** properties can be used to obtain the size of the image. The printer gets initialized after a **StartDoc** command, or a **Lprint** command (invokes **StartDoc** and **StartPage** implicitly).

Example

```
OpenW 1
AutoRedraw = 1
Local h As Handle, n As Int32
For n = 1 To 601 Step 100 : Line 0, n, 601, n :
  Line n, 0, n, 601 : Next n
' A StartDoc is only necessary for
' PrintFormWidth and PrintFormHeight.
Dlg Print Win_1, 0, h
If h <> 0
  SetPrinterHDC h
  Printer.StartDoc "Text"
  Trace Win_1.PrintFormHeight
  Trace Win_1.PrintFormWidth
  Trace Win_1.PrintFormHeight(1)
  Trace Win_1.PrintFormWidth(1)
  Me.PrintForm 1
  Printer.EndDoc
  Debug.Show
EndIf
CloseW 1
```

Remarks

The **PrintForm** method creates a Picture object from the Form. This Picture object is also obtainable using the **PrintPicture** or **PrintPicture2** properties.

See Also

Form, Printer, PrintPicture

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

PrintPicture, PrintPicture2 Properties

Purpose

Returns the **Picture** object created with **PrintForm**.

Syntax

Set p1 = Form.**PrintPicture**

Set p1 = Form.**PrintPicture2**

p1:Picture Object

Description

The **PrintForm** method creates a Picture object from the Form. This Picture object is also obtainable using the **PrintPicture** or **PrintPicture2** properties. **PrintPicture** returns an image of the client area with all visible objects and bitmaps of the **Form** object. **PrintPicture2** returns the same, including the window borders.

Example

```
Global Picture p1, p2 : Global h As Handle
OpenW 1 : AutoRedraw = 1
Color QBColor(2) : PCircle 200, 200, 100, 35, 220
Set p1 = Me.PrintPicture
Set p2 = Me.PrintPicture2
Dlg Print Win_1, 0, h
If h <> 0
```

```
  SetPrinterHDC h
  Output = Printer
  Printer.StartDoc "Test"
  Printer.StartPage
  PaintPicture p1, 0, 0
  Printer.NewFrame
  PaintPicture p2, 0, 0
  Printer.EndPage
  Printer.EndDoc
  Output = Me
EndIf
CloseW 1
```

This prints the form in the size of a stamp. Use the other parameters of PaintPicture to scale the bitmap.

See Also

Form, Printer, PrintForm, Lprint, StartDoc

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

SysMenuText Property, SysMenuOver Event

Purpose

Returns or sets the text displayed for the system menu of a Form.

Syntax

Form.**SysMenuText**(idx%) [= txt]

Sub Form_**SysMenuOver**([index%,] idx%)

txt:sexp
index%:iexp, Form number
idx%:iexp, menu item identifier

Description

The *window menu* (also known as the *System menu* or *Control menu*) is a pop-up menu defined and managed almost exclusively by the operating system. The user can open the window menu by clicking the application icon on the title bar or by right-clicking anywhere on the title bar.

The window menu provides a standard set of menu items that the user can choose to change a window's size or position, or close the application. Items on the window menu can be added, deleted, and modified, but most applications just use the standard set of menu items.

The window menu initially contains items with various identifier values, such as SC_CLOSE, SC_MOVE, and SC_SIZE. These command identifiers are used as the parameter in **SysMenuText**() to modify the text.

When the mouse hovers over the items in the window menu, GFA-BASIC 32 invokes the **SysMenuOver** event passing the identifier in the *idx%* argument.

This parameter can be one of the following values:

SC_CLOSE	Closes the window.
SC_CONTEXTHELP	Changes the cursor to a question mark with a pointer. If the user then clicks a control in the dialog box, the control receives a WM_HELP message.
SC_DEFAULT	Selects the default item; the user double-clicked the **window** menu.
SC_HOTKEY	Activates the window associated with the application-specified hot key. The low-order word of *lParam* identifies the window to activate.
SC_HSCROLL	Scrolls horizontally.
SC_KEYMENU	Retrieves the **window** menu as a result of a keystroke.
SC_MAXIMIZE	Maximizes the window.
SC_MINIMIZE	Minimizes the window.
SC_MONITORPOWER	Sets the state of the display. This command supports devices that have power-saving features, such as a battery-powered personal computer.

	lParam can have the following values: 1 means the display is going to low power. 2 means the display is being shut off.
SC_MOUSEMENU	Retrieves the **window** menu as a result of a mouse click.
SC_MOVE	Moves the window.
SC_NEXTWINDOW	Moves to the next window.
SC_PREVWINDOW	Moves to the previous window.
SC_RESTORE	Restores the window to its normal position and size.
SC_SCREENSAVE	Executes the screen saver application specified in the [boot] section of the SYSTEM.INI file.
SC_SIZE	Sizes the window.
SC_TASKLIST	Activates the **Start** menu.
SC_VSCROLL	Scrolls vertically.

All predefined **window** menu items have identifier numbers greater than 0xF000. If an application adds commands to the **window** menu, it should use identifier numbers less than 0xF000.

Example

```
OpenW 1, 10, 10, 300, 300
Ocx StatusBar stb
Win_1.SysMenuText(SC_MINIMIZE) = "Hello"
Do
  Sleep
Until Win_1 Is Nothing
```

```
Sub Win_1_SysMenuOver(idx%)
  If idx% = 0
    stb.SimpleText = ""
  Else
    stb.SimpleText = "System menu idx = 0x" &
      Hex(idx)
  EndIf
EndSub
```

Replaces Minimize by Hello and shows the identifier in the status bar.

See Also

Form

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

Validate, ValidateAll Method

Purpose

Validates the client area of a **Form** within a rectangle by removing the rectangle from the update region of the specified window.

Syntax

Form.**Validate** [left],[top],[width],[height]

Form.**ValidateAll**

left, top, width, height:Single exp

Description

Validate is used to prevent redrawing of a rectangle. The upper left corner of the rectangle is given in *left* and *top*, the width in *width* and the height in *height*.

ValidateAll is used to prevent redrawing of the entire client-area rectangle.

Remarks

Validate suppresses a redraw message aimed at a specific rectangle. **Validate** corresponds to the Windows function *ValidateRect*().**ValidateAll** corresponds to the Windows function *ValidateRect*(,Null).

Example

See Also

Form

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

WhatsThisMode Method

Purpose

Causes the mouse pointer to change into the What's This pointer and prepares the application to display What's This Help on the selected object.

Syntax

Form.**WhatsThisMode**

Description

Executing the **WhatsThisMode** method places the application in the same state you get by clicking the What's This button in the title bar. The mouse pointer changes to the What's This pointer. When the user clicks an object, the WhatsThisHelpID property of the clicked object is used to invoke context-sensitive Help. This method is especially useful when invoking Help from a menu in the menu bar of your application.

Example

```
OpenW 1
Ocx Command cmd = "Activate WhatsThis Mode", 10,
 10, 140, 22
Do : Sleep : Until Win_1 Is Nothing

Sub cmd_Click
 Win_1.WhatsThisMode
EndSub
```

For a fuller example dealing with **WhatsThisHelpID**, see Form

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

MessageProc, DDEWndProc Events

Purpose

Call back procedures for window messages of a Form.

Syntax

Sub *Form*_**MessageProc**(hWnd%, Mess%, wParam%, lParam%, retval%, ValidRet?)

Sub *Form*_**DDEWndProc**(hWnd%, Mess%, wParam%, lParam%, retval%, ValidRet?)

Description

With the **MessageProc** event sub you can actually filter or influence the behaviour of the form/window procedure for the Form window class. The **MessageProc** is called before gb32 handles the message itself. The sub obtains six parameters. The first four are the same as defined for any Windows API window procedure: *hWnd%, Mess%, wParam%,* and *lParam%*. Every message, whether it is obtained from the message queue (the posted messages) or by a direct call from any other source (SendMessage), is handled in the window procedure. The *hWnd* parameter is the window to which the message is sent (because the MessageProc is attached to a Form, we already know the objects name) The *Mess* parameter is the message number- which is usually a constant such as WM_ACTIVATEAPP or WM_PAINT. The *wParam* and *lParam* parameters differ for each message, as does the return value; you must look up the specific message to see what they mean. Often, *wParam* and the return value is ignored, but not always.

The **MessageProc** event sub has two additional parameters (ByRef) that allow you to return a value. For instance, when you don't want GB32 to handle a certain message you can set the *ValidRet?* variable to True and provide a return value by setting *RetVal%*. What value *RetVal* must have is defined in the Windows API SDK. It often says something like: "If you handle this message return zero (or..)".

The **DDEWndProc** is a call back procedure as well. It is invoked from inside the window procedure for the form/window. However, the **DDEWndProc** is only invoked for DDE messages: WM_DDE_ACK, WM_DDE_POKE, WM_DDE_EXECUTE, WM_DDE_DATA, WM_DDE_ADVISE, WM_DDE_UNADVISE, or WM_DDE_INITIATE, and WM_DDE_REQUEST. The *RetVal%* and *ValidRet?* variables are used to return values.

Example

Now let us look at an example. Suppose you want to store the window coordinates of **OpenW** #1 in the register so the application can use these value to open at the same place. In GB32 you could handle the sub events **ReSize** and **Moved** to store the coordinates. As an alternative, you could use **MessageProc** and handle the WM_EXITSIZEMOVE message.

```
Sub Win_1_MessageProc(hWnd%, Mess%, wParam%,
 lParam%, Retval%, ValidRet?)
  Local Int x, y, w, h
  Switch Mess
  Case WM_EXITSIZEMOVE
```

```
    GetWinRect hwnd, x, y, w, h
    SaveSetting "MyComp", "ThisApp", "Position",
     Mkl$(x, y, w, h)
    ValiRet? = True
    RetVal = 0
  EndSwitch
EndSub
```

Remarks

The **MessageProc** event sub actually _is_ the subclass procedure for the GB32 **OpenW**, **Form,** and **Dialog** windows. Subclassing is a built-in feature of GFA-BASIC 32. The OCX Form is perfectly suited to write custom controls.

When **OpenW** uses a number > 31, the window is accessed using **Form**(n) and the event subs as Form_event(index%, …).

See Also

Form, MessageE

{Created by Sjouke Hamstra; Last updated: 17/10/2014 by James Gaite}

DisplayChange, SysColorChange, WinIniChange Events

Purpose

These events occur when a system setting changes.

Syntax

Sub *Form*_**DisplayChange** [(*index%*)]

Sub *Form*_**SysColorChange** [(*index%*)]

Sub *Form*_**WinIniChange** [(*index%*)]

Description

The **DisplayChange** event occurs when the display resolution has changed. The new image depth of the display in bits per pixel can be obtained with **_C**. The **Screen.cxScreen** specifies the new horizontal resolution of the screen. The **Screen.cyScreen** property specifies the new vertical resolution of the screen.

The **SysColorChange** event occurs when a change is made to a system color setting. The system sends a WM_PAINT message to any window that is affected by a system color change. Applications that have brushes using the existing system colors should delete those brushes and recreate them using the new system colors.

The **WinIniChange** event occurs when the *SystemParametersInfo* function changes a system-wide setting. The system sends this message only if the *SystemParametersInfo* caller specifies the SPIF_SENDCHANGE flag.

Example

```
Sub Win_1_DisplayChange
  Print _C , Screen.cxScreen, Screen.cyScreen
EndSub

Sub Win_1_WinIniChange
  Print Screen.WorkLeft; Screen.WorkTop; _
    Screen.WorkWidth; Screen.WorkHeight
EndSub
```

See Also

Form

{Created by Sjouke Hamstra; Last updated: 30/09/2014 by James Gaite}

EndSession, QueryEndSession Events

Purpose

The **EndSession** event informs the application whether the session is ending. The **EndSession** event occurs after the system processes the results of the **QueryEndSession** event. **QueryEndSession** event occurs when the user chooses to end the session or when an application calls the *ExitWindows* function.

Syntax

Sub *Form_* **QueryEndSession**(Cancel?)

Sub *Form_* **EndSession**

Description

The WM_QUERYENDSESSION message, which is responsible for the **QueryEndSession** event, is sent when the user chooses to end the Windows session or when an application calls the *ExitWindows* function. If any application returns zero, the session is not ended. The system stops sending WM_QUERYENDSESSION messages as soon as one application returns halts the process. To prevent the system from ending the session, set the Cancel? variable of the **QueryEndSession** event to True.

After processing this message, the system sends the WM_ENDSESSION message, which leads to the **EndSession** event. The **EndSession** event sub should be as clean as possible, graphics output is not possible and file I/O should be minimized.

Cleaning up should be performed in the **QueryEndSession** event.

Example

```
OpenW 1
Do
  Sleep
Until Me Is Nothing

Sub Win_1_QueryEndSession(Cancel?)
  MsgBox "Save data?"
EndSub

Sub Win_1_EndSession
EndSub
```

Remarks

Windows doesn't send WM_CLOSE and WM_DESTROY messages when the user logs off. WM_QUERYENDSESSION is the time to do the final things.

See Also

Form, Close, Destroy

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

HScroll, HScrolling, VScroll, VScrolling Events

Purpose

These events occur when a horizontal or vertical **Form** scrollbar is scrolling the scroll box or after a scroll event.

Syntax

Sub Form_**HScroll** [(index%)]

Sub Form_**HScrolling** [(index%)]

Sub Form_**VScroll** [(index%)]

Sub Form_**VScrolling** [(index%)]

Description

The **HScroll** event occurs when a scroll event occurs in the window's standard horizontal scroll bar. The **HScrolling** event occurs when a user is scrolling the scroll box in the window's standard horizontal scroll bar.

The **VScroll** event occurs when a scroll event occurs in the window's standard vertical scroll bar. The **VScrolling** event occurs when a user is scrolling the scroll box in the window's standard vertical scroll bar.

A scroll event is invoked only when the scrollbars are visible. Use the **Scrollbars** property to determine which scrollbars to set:

basNoScroll - None

basHorizontal - Horizontal scrollbar only.

basVertical - Vertical scrollbar only

basBoth - Both

This property can be set at run time and at design time.

Example

```
Debug.Show
OpenW 100
Form(100).ScrollBars = basBoth
Do
  Sleep
Until Me Is Nothing

Sub Form_HScrolling(index%)
  Trace Me.HScTrack
EndSub

Sub Form_HScroll(index%)
  Trace Me.HScPos
EndSub

Sub Form_VScrolling(index%)
  Trace Me.VScTrack
EndSub

Sub Form_VScroll(index%)
  Trace Me.VScPos
EndSub
```

See Also

Form, Scrollbars, VScMax, VScMin, VScPos, VScPage, VScStep, VScTrack, HScMax, HScMin, HScPos, HScPage, HScStep, HScTrack

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

McіNotify Event

Purpose

Occurs when the MM_MCINOTIFY message is received form a Mci device.

Syntax

Sub Form_**MciNotify**(devID%, Code%)

Description

Use this event to handle the MM_MCINOTIFY message ($3b9).

The *Code*% is returned in wParam.

wParam=1 - Mci command aborted

wParam=2 - Mci command successful

wParam=4 - Mci superseded by a new notify command

wParam=8 - Mci error, not reported when using **Mci$**()

The devID% is the device ID (devID%) sending the message, it is returned in the loword of the lParam.

LoWord(lParam) = Device ID

Example

```
OpenW 1
Local t As Double = Timer
```

```
Mci "open c:\windows\media\alarm01.wav alias bong"
~Mci$("play bong from 1 notify")
Do
  PeekEvent
  Print AT(1, 1); "Playing Track: "; Format(Timer -
    t, "0.000"); " secs"
Loop Until Mci$("status bong mode") != "playing"
Mci "close bong"
CloseW 1

Sub Win_1_MciNotify(devID%, Code%)
  Debug.Show
  Trace Code%
  Trace devID%
  Trace mciID("bong")
EndSub
```

See Also

Form, Mci$, Mci

{Created by Sjouke Hamstra; Last updated: 16/10/2014 by James Gaite}

MonitorPower, ScreenSave Events

Purpose

Occurs when a screensaver starts or monitor goes to low power.

Syntax

Sub *Form*_**MonitorPower**(lParam%, Cancel?)

Sub *Form*_**ScreenSave**(Cancel?)

Description

The **ScreenSave**(Cancel?) occurs when the screensaver is starting. Set Cancel? = True to prevent the start of the screensaver.

The **MonitorPower**(lParam%, Cancel?) occurs when the display is going to low-power. The *lParam* can have the following values:

1 means the display is going to low power.

2 means the display is being shut off.

Set Cancel? = True to prevent monitor power mode.

See Also

Form

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

MouseDblClick Event

Purpose

Occur when the user double clicks a mouse button in a **Form**.

Syntax

Sub Form**_MouseDblClick(**[index%,] *button&*, *shift&*, *x!*, *y!***)**

Form:Form Object
button&, shift&:Short integer exp
x!, y!:Single exp

Description

Object - Returns an **Form** Ocx object expression.

index% - Returns an integer that uniquely identifies a **Form** if it's in an array.

button - Returns an integer that identifies the button that was pressed. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.

shift - Returns an integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released. A bit is set if the key is down. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of shift would be 6.

x, y - Returns a number that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Use a **MouseDblClick** event for a better response of a double click event in a Form. Unlike the **DblClick** event, the **MouseDblClick** event enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Example

```
OpenW # 1
Do
  Sleep
Until Me Is Nothing

Sub Win_1_MouseDblClick(Button&, Shift&, x!, y!)
  Print "Mouse Double Click - Button: "; Button&
EndSub

Sub Win_1_DblClick()
  Print "Double Click"
EndSub
```

Remarks

The following applies to both **Click** and **DblClick** events:

·If a mouse button is pressed while the pointer is over a form or control, that object "captures" the mouse and receives all mouse events up to and including the last **MouseUp** event. This implies that the x, y mouse-pointer coordinates returned by a mouse event may not always be in the internal area of the object that receives them.

·If mouse buttons are pressed in succession, the object that captures the mouse after the first press receives all mouse events until all buttons are released.

See Also

Form, Click, DblClick, MouseMove

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

MouseWheel Event

Purpose

Occurs when the user moves the mousewheel in a **Form**.

Syntax

Sub Form**_MouseWheel(**[index,] *Buttons&, Delta%, MseX%, MseY%***)**

Form:Form Object
Buttons&:Short integer exp
Delta%, MseX, MseY:iexp

Description

FormReturns a **Form** object expression.

index - Returns an integer that uniquely identifies a Form if it's in a Form() array.

Button - Indicates whether various virtual keys are down. This parameter can be any combination of the following values:

MK_CONTROL	Set if the ctrl key is down.
MK_LBUTTON	Set if the left mouse button is down.
MK_MBUTTON	Set if the middle mouse button is down.
MK_RBUTTON	Set if the right mouse button is down.
MK_SHIFT	Set if the shift key is down.

Delta - Indicates the distance that the wheel is rotated, expressed in multiples or divisions of WHEEL_DELTA, which is 120. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user.

MseX, MseY - Returns a number that specifies the current location of the mouse pointer in pixels - this is relative to the desktop rather than the window/form over which the mouse is hovering.

Example

```
Global Int x = ((Screen.x - 100) / 2), y =
  ((Screen.y - 14) / 2)
OpenW Full 1
Ocx Label lb = "***---***", x, y, 100, 14 :
  lb.Alignment = 2
Do
  Sleep
Until Me Is Nothing

Sub Win_1_MouseWheel(Buttons&, Delta%, MseX%,
  MseY%)
  If Buttons& = 0                    // Vertical
    movement
    y = y - (Delta% / 4)
  Else If Buttons& = MK_SHIFT    // Horizontal
    movement
    x = x - (Delta% / 4)
  EndIf
  // Keep label in-screen (although it does go
    behind the taskbar)
  x = Max(0, Min(x, Screen.x - 100))
  y = Max(0, Min(y, Screen.y - 14 -
    Screen.cyCaption))
  lb.Move x, y
EndSub
```

Remarks

See Also

Form, Click, DblClick, MouseDown

{Created by Sjouke Hamstra; Last updated: 02/03/2018 by James Gaite}

ReSize and Moved Event

Purpose

The **ReSize** event occurs when the window state of a **Form** changes. (For example, a form is maximized, minimized, or restored.)

The **Moved** event occurs when a **Form** is moved or being moved to a new position or a Form's Top or Left property settings have been changed programmatically.

Syntax

Sub Form_**Resize**([index%])

Sub Form_**Moved**([index%])

index%:iexp, Form number

Description

These events occurs when an object is sized, (being) moved to a new position, or when the settings for the **Top**, **Left**, **Width**, or **Height** properties have been changed in code.

Example

```
Debug.Show
OpenW 1, 10, 10, 300, 300
Do
  Sleep
Until Win_1 Is Nothing
```

```
Sub Win_1_ReSize
  Debug "Resize"

Sub Win_1_Paint
  Debug "Paint"

Sub Win_1_Moved
  Debug Me.Left' in twips
  Debug Me.Top' in twips
```

Remarks

Use a **ReSize** event procedure to move or resize controls when the parent form is resized. You can also use this event procedure to recalculate variables or properties, such as **ScaleHeight** and **ScaleWidth** that may depend on the size of the form. If you want graphics to maintain sizes proportional to the form when it's resized, use the **Paint** event, which follows the **ReSize** event.

For a Form, the **Left** and **Top** properties are in twips. For Ocx controls in the client the new cooridnates are in **ScaleMode** units if **OcxScale** = 1.

See Also

Form

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

OnHelp, OnCtrlHelp, OnMenuHelp Events (Form)

Purpose

These **Form** events occur when F1 is pressed or when the What's This mouse cursor [?] is clicked.

Syntax

Sub *Form*_**OnHelp**(*[index%,] Flg%, ID%, hWnd%, Ctx%, x%, y%*)

Sub *Form*_**OnCtrlHelp**(*[index%,] Ctrl As Object, x%, y%*)

Sub *Form*_**OnMenuHelp**(*[index%,] idx%, x%, y%*)

Description

These events occur when F1 is pressed or when the What's This mouse cursor [?] is clicked. The **OnHelp** event is called for non-Ocx objects and **OnCtrlHelp** for Ocx objects. The **OnMenuHelp** event occurs when the mouse is over a menu entry and F1 is pressed. It is also possible to use an on screen control or program code to switch on the What's This mouse cursor by setting the the WhatsThisMode property of the window itself.

If the help is required for an Ocx child window, the Form receives an **OnCtrlHelp** event specifying the Ocx control and the mouse coordinates in the arguments of the event. To identify the help associated with the Ocx object set the

WhatsThisHelpID or the **HelpContextID** properties with a value.

When the help is required for a normal control the **OnHelp** event is executed passing the following arguments:

Flg%	Type of context for which Help is requested. This can be one of HELPINFO_MENUITEM - Help requested for a menu item. HELPINFO_WINDOW - Help requested for a control or window.
ID%	Identifier of the child window or control.
hWnd%	Window handle of the control.
Ctx%	Help context identifier of the window or control set with *SetWindowContextHelpId(h, id)* API function.
x%, y%	The screen coordinates of the mouse cursor. This is useful for providing Help based on the position of the mouse cursor.

To identify the help associated with a control object set a value using *SetWindowContextHelpId(h, id)* API function (equivalent to the **WhatsThisHelpID** or the **HelpContextID** properties).

When the mouse is over a menu entry and F1 is pressed, the **OnMenuHelp** event sub is invoked, identifying the currently selected menu item in *idx%* and the mouse coordinates in *x%* and *y%*.

To show the relevant help page in a WinHelp (.hlp) file use the ShowHelp; for a HTMLHelp (.chm) file, see Accessing HTML Help Files.

Example

```
// If you a calling the WinHelp (.hlp) file, set
  winhelp? to True
Dim winhelp? = False
OpenW 1
Win_1.MinButton = False : Win_1.MaxButton = False
  : Win_1.HelpButton = True
If winhelp?
  Ocx CommDlg cd : cd.HelpFile = "gfawin32.hlp"
Else
  Declare Function HTMLHelpTopic Lib "hhctrl.ocx"
    Alias "HtmlHelpA" (ByVal hwndCaller As Long,
    ByVal pszFile As String, ByVal uCommand As
    Long, ByVal dwData As String) As Long
  Global helpdir$ =
    GetSetting("\\HKEY_CLASSES_ROOT\Applications\Gf
    aWin32.exe\shell\open\command", , "")
  helpdir$ = Left(helpdir$, RInStr(helpdir$, "\"))
    & "GFAWin32.chm" : If Left(helpdir$, 1) = #34
    Then helpdir$ = Mid(helpdir$, 2)
  Global Const HH_DISPLAY_INDEX = &H2
EndIf
Dim m$()
Array m$() = "File"#10 "Exit"#10#10
Menu m$()
Ocx Command cmd = "Push", 10, 10, 80, 24
cmd.WhatsThisHelpID = 1002
PushButton "Button", 100, 10, 40, 80, 24
// Give a normal control a WhatsThisHelpID:
~SetWindowContextHelpId(Dlg(-1, 100), 1003)
LocaXY 1, 10
Do
  Sleep
Until Me Is Nothing

Sub Win_1_OnCtrlHelp(Ctrl As Object, x%, y%)
```

```
  Debug.Print "OnCtrlHelp: WhatsThisHelpID = ";
    Ctrl.WhatsThisHelpID
  If winhelp?
    cd.HelpContext = Ctrl.WhatsThisHelpID
    cd.HelpCommand = cdhContext
    cd.ShowHelp
  Else
    HTMLHelpDisplay(Ctrl.WhatsThisHelpID)
  EndIf
EndSub

Sub Win_1_OnHelp(Flg%, ID%, hWnd%, Ctx%, x%, y%)
  Debug.Print "OnHelp: Ctx = "; Ctx
  If winhelp?
    cd.HelpContext = Ctx
    cd.HelpCommand = cdhContext
    cd.ShowHelp
  Else
    HTMLHelpDisplay(Ctx%)
  EndIf
EndSub

Sub Win_1_OnMenuHelp(idx%, x%, y%)
  Debug.Print "OnMenuHelp: idx% = "; idx `
    Me.MenuItem(idx).Text
  If winhelp?
    cd.HelpContext = idx%
    cd.HelpCommand = cdhContext
    cd.ShowHelp
  Else
    HTMLHelpDisplay(idx%)
  EndIf
EndSub

Sub HTMLHelpDisplay(helpvalue%)
  Local HHhWnd As Int32
  // At the time of writing, this help file does
    not have ContextIDs
  // Use the returned WhatsThisHelpID as a pointer
    as below
  Select helpvalue%
  Case 1
    HHhWnd = HTMLHelpTopic(Null, helpdir$,
      HH_DISPLAY_INDEX, "menu")
  Case 1002
    HHhWnd = HTMLHelpTopic(Null, helpdir$,
      HH_DISPLAY_INDEX, "command")
  Case 1003
    HHhWnd = HTMLHelpTopic(Null, helpdir$,
      HH_DISPLAY_INDEX, "pushbutton")
  EndSelect
  ~SetForegroundWindow(HHhWnd) : SendKeys #13
EndSub
```

See Also

Form, ShowHelp

{Created by Sjouke Hamstra; Last updated: 17/07/2015 by James Gaite}

Form Command

Purpose

Creates a (MDI) form.

Syntax

Form [options] fname [= [title$],[x],[y],[w, h]]

Form [options] **MdiParent** fname [= [title$],[x],[y],[w, h]]

Form [options] **MdiChild Parent** form, fname [= [title$],[x],[y],[w, h]]

Form [options] **Owner** form, fname [= [title$],[x],[y],[w, h]]

options: [Tool] [Center] [Full] [Hidden] [Client3D] [Help] [Top] [Palette] [NoCaption] [NoTitle] [Fixed][Default]

fname, form:Form Object variable
title$:sexp, optional
x, y:iexp, optional
w, h:iexp, optional

Description

A **Form** is a window or dialog box that makes up part of an application's user interface. The **Form** command creates a **Form** object with the specified *name*. The name is used in code to identify the form. The *name* property must start with a letter and can be a maximum of 40 characters. It can include numbers and underline (_) characters but cannot include punctuation or spaces.

The *options* argument specifies additional window state settings.

Center centers the form.

Full creates a maximized window, excludes **Hidden** (full windows are always visible).

Hidden opens invisible

Client3D sets WS_EX_CLIENTEDGE

Tool creates a WS_EX_TOOLWINDOW

Help includes a Help button in the window caption, excludes minimize an maximize buttons

Top creates a topmost window

Palette creates a WS_EX_PALETTEWINDOW

Fixed a non-sizable window

NoCaption no title bar

NoTitle no title bar, alias

Default uses Windows default values

The **Form** command can also be used to create MDI parent and child windows.

Form [options] **MdiParent** *form* creates a parent MDI window (like **ParentW**).

Form [options] **MdiChild Parent** form, *name* creates a MDI child window *name* of MDI parent *form* (like **ChildW**).

```
Form MdiParent test  = , , 20 , 300 , 300
Form MdiChild Parent test, ch2
Form Hidden MdiChild Parent test, ch1 = "ChildW
 ch1", , , 10, 10
```

An **Ocx Form** is control with all the attributes of a Form. An **Ocx Form** is used as a child form inside a parent form. **Ocx Form** is equivalent to VB's PictureBox.

Example

```
Form ftest = "Test Form", , , 300, 300
// to create a form
Local a%
Print "GFA-BASIC 32"
Print
Print "Press any key"
KeyGet a%
ftest.Close
//or
Form ftest = "GFA", 10, 10, 200, 300
// center it in the middle of the desktop
ftest.Center 0
// OpenW 1, 10, 10, 300, 400
// if used for a windows with it's handle
// Win_1.hWnd
// test.Center Win_1.hWnd
Print "Press any key"
KeyGet a%
ftest.Close
```

Remarks

OpenW #n, **ChildW** #n, and **ParentW** #n are commands that create a Form, whose name is predestined by GFA-BASIC 32. These commands take a number n in the range from 0 to 31 to be identified by. These commands get the Form name **Win_***n*, where n is the window number (**Win_0** .. **Win_31**). A value greater than 31 will provide the window with the Form object name **Form**(n).

The same is true for the Dialog command, which takes a number from 0 to 31 as well. The Form object for the dialog boxes is **Dlg_0** .. **Dlg_31.**

For an example of **ParentW**, **ChildW**, and **Ocx Form** see **ParentW**.

See Also

Form Object, OpenW, ChildW, ParentW, Dialog

{Created by Sjouke Hamstra; Last updated: 06/10/2014 by James Gaite}

DayBold, DayVisible, VisibleDays Properties, GetDayBold Event (MonthView)

Purpose

DayBold returns or sets a value that determines if a displayed day is bold. **DayVisible** returns a Boolean indicating whether the date is visible. The **VisibleDays** property returns an array containing the dates that are currently visible.

Syntax

MonthView.**DayBold**(*date*) [= *Boolean*]

MonthView.**DayVisible**(*date*)

MonthView.**VisibleDays**(*index%*)

Sub *MonthView*_**GetDayBold**(*StartDate As Date, Count%, State?()*)

Description

The *date* parameter in the **DayBold**(*date*) property specifies a date found in the **VisibleDays** property and **DayBold** specifies whether or not the date is bolded (True).

The **DayBold** property is an array that corresponds to the **VisibleDays** property. Each Boolean element indicates whether its corresponding date should be displayed in bold. Only dates that are currently displayed are valid. Valid dates can be found by looking in the **VisibleDays** property.

The *index* parameter in the **VisibleDays**(*index*) is an integer which specifies a displayed date on the calendar. *Index* can be any value from 1 to 41. A value of 1 indicates the first date that is currently displayed.

Only dates that are currently displayed can be found in the **VisibleDays** property. In addition, the number of visible days can changes depending on the settings of the **MonthColumns** and **MonthRows** properties. As you move from month to month, the information in this property is not preserved.

The **DayVisible**(*date*) property returns a Boolean indicating if the specified date is currently visible in the MonthView Ocx.

The **GetDayBold** event occurs when the control needs to display a date, in order to get bold information. The event can be used to set the boldness of days as they are brought into view. The event has three parameters. *StartDate* specifies the first date that is displayed, *count* the number of days that are displayed, and *State?()* is an array of Boolean values that specify if a date is bold.

Example

```
OpenW 1, , , 260, 220
Ocx MonthView mvw = "", 10, 10, 0, 0 /* Width and
 Height are ignored
.DayBold(.VisibleDays(1)) = True
.DayBold(.VisibleDays(41)) = True
Print .DayVisible(Date)
```

```
Local n
For n = 1 To 41 : Debug mvw.VisibleDays(n) : Next
  n
Do : Sleep : Until Win_1 Is Nothing

Sub mvw_GetDayBold(StartDate As Date, Count%,
  State?())
```

See Also

MonthView

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

DayOfWeek, StartOfWeek, Week Properties

Purpose

DayOfWeek returns or sets a value that specifies the current day of week. **StartOfWeek** specifies the starting day of the week. **Week** specifies the current week number.

Syntax

MonthView.**DayOfWeek** [= *number*]

MonthView.**StartOfWeek** [= *number*]

MonthView.**Week** [= *number*]

Description

The **DayOfWeek** property specifies the day of the week (1 to 7). Sunday = 1, Monday = 2, etc.

The **StartOfWeek** property specifies the starting day of the week (1 to 7). Sunday = 1, Monday = 2, etc.

The **Week** property evaluates to an integer indicating the week number (1 to 52).

Example

```
OpenW 1, 100, 100, 260, 220
Ocx MonthView mvw = "", 10, 10, 0, 0
mvw.StartOfWeek = 1 ' Sunday
OpenW 2, 400, 100, 200, 300
```

```
Do : Sleep  : Until Win_1 Is Nothing
CloseW 2

Sub mvw_MouseUp(Button&, Shift&, x!, y!)
  Local a$ = "Week No: " & mvw.Week & "  Day of
    Week: " & mvw.DayOfWeek
  Set Me = Win_2 : Print a$ : Set Me = Win_1
EndSub

Sub Win_2_Close(Cancel?)
  CloseW 1
EndSub
```

See Also

MonthView

{Created by Sjouke Hamstra; Last updated: 28/09/2014 by James Gaite}

MaxDate, MinDate, Today, Value Properties

Purpose

Returns or sets the first and last date allowed by the calendar. **Today** sets a new current date, and **Value** returns the current selection.

Syntax

MonthView.**MaxDate** [**=** *date*]

MonthView.**MinDate** [**=** *date*]

MonthView.**Value** [= *date*]

date = *MonthView*.**ToDay**

Description

MinDate and **MaxDate** return or set the minimum and maximum for the **Value** property for the specified control. The **Value** property returns or sets the current date of the control. The **Value** property is the default property of the control.

Today retrieves the date for the date specified as "today".

Example

```
OpenW Fixed 1, 10, 10, 225 + (Screen.cxFixedFrame
  * 2), 159 + (Screen.cyCaption +
  (Screen.cyFixedFrame * 2))
```

```
Ocx MonthView mvw
mvw.MinDate = #01.07.1998#
mvw.MaxDate = #01.07.1999#
mvw.Value = #17.04.1999#
OpenW Fixed 2, 260, 10, 225 + (Screen.cxFixedFrame
  * 2), 159 + (Screen.cyCaption +
  (Screen.cyFixedFrame * 2))
Ocx MonthView mvw2
Debug.Show
~SetWindowPos(Debug.hWnd, 0, 510, 5, 400, 300, 0)
Trace mvw2.MinDate
Trace mvw2.MaxDate
Trace mvw2.Value
Trace mvw2.Today
Do : Sleep : Until (Win_1 Is Nothing) Or (Win_2 Is
  Nothing)
CloseW 1 : CloseW 2 : Debug.Hide

Sub mvw2_MouseUp(Button&, Shift&, x!, y!)
  Debug.Print
  Trace mvw2.Value
EndSub
```

Remarks

If the **Today** selection is set to any date other than the default, the following conditions apply:

- The control will not automatically update the "today" selection when the time passes midnight for the current day.

- The control will not automatically update its display based on locale changes.

See Also

MonthView

{Created by Sjouke Hamstra; Last updated: 15/10/2014 by James Gaite}

MultiSelect, MaxSelCount, SelEnd, SelStart Properties (MonthView)

Purpose

MultiSelect returns or sets a value that determines if multiple dates can be selected at once. **MaxSelCount** returns or sets the maximum number of contiguous days that can be selected at once. **SelEnd** and **SelStart** returns or sets the upper and lower bounds of the date range that is selected.

Syntax

object.**MultiSelect** [= *boolean*]

object.**MaxSelCount** [= *number%*]

object.**SelEnd** [= *date*]

object.**SelStart** [= *date*]

object: MonthView
date:Date exp

Description

The **MultiSelect** property allows the user to select multiple days (True = Default). When set to False, the user is not allowed to select multiple days. By default, the control allows the user to select a range of dates. The default maximum range is one week (7 days). You can change the maximum selectable range by setting the **MaxSelCount** property. The **Value** property will be in this range, indicating which date has focus.

The **MaxSelCount** property is valid only when the **MultiSelect** property is to **True**. Additionally, the **MaxSelCount** property must be set to a value that is greater than the difference between the **SelStart** and **SelEnd** properties. For example, given a selection of 9/15 to 9/18, MonthView.SelEnd - MonthView.SelStart = 3. However, four days are actually selected; thus **MaxSelCount** must be set to 4. The default of the property is one week (7 days).

The **SelStart** property defines the lower bound of the date range that is selected. The **SelEnd** property defines the upper bound of the date range that is selected.

The range of selected dates can span multiple months. It can include dates that are not currently displayed.

In order for multiple date selection to work properly, the **MaxSelCount** property must be set to a value that is greater than the difference between the **SelStart** and **SelEnd** properties.

The **SelStart** and **SelEnd** settings are only valid if the **MultiSelect** property is set to **True**. In addition, if the date range you are trying to select is not visible then an error will be raised - to get around this, set **Value** first to show the required month (or months if you have more than one shown) and then enter the values to define the required selection.

Example

```
// This example highlights the week encompassing
  the selected date...
// ...running from Sunday to Saturday
Ocx MonthView mvw : mvw.MultiSelect = True
Do  : Sleep : Until Me Is Nothing

Sub mvw_MouseUp(Button&, Shift&, x!, y!)
  Local dateclicked As Date = mvw.Value, wd| =
    WeekDay(dateclicked), se As Date, ss As Date
  se = DateAdd("d", -(wd| - 1), dateclicked) //
    Find preceding Sunday
  ss = DateAdd("d", (7 - wd|), dateclicked)  //
    Find next Saturday
  Trace wd| : Trace dateclicked : Trace se : Trace
    ss
  mvw.Value = ss : mvw.SelStart = ss : mvw.SelEnd =
    se
EndSub
```

See Also

MonthView

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

TitleBackColor, TitleForeColor, TrailingForeColor, MonthBackColor Properties (MonthView)

Purpose

These properties returns or sets a value that specifies the color displayed within a several areas of the **MonthView** Ocx control.

Syntax

MonthView.**TitleBackColor** [= *color*]

MonthView.**TitleForeColor** [= *color*]

MonthView.**TrailingForeColor** [= *color*]

MonthView.**MonthBackColor** [= *color*]

color:iexp; RGB color

Description

The **TitleBackColor** and **TitleForeColor** properties specify the background and foreground colors of the title area of the control.

The **TrailingForeColor** property determines the color of trailing dates. Trailing dates are day numbers that are displayed which precede and follow day numbers of the currently selected month. By default, trailing dates are displayed in white.

The **MonthBackColor** property determines the background color displayed within a month.

On later versions of Windows (especially Windows 8), these properties have no effect as they are fixed by the OS.

Example

```
// On later versions of Windows, these properties
 have no effect
Ocx MonthView mvw
With mvw
 .TitleBackColor = QBColor(2)
 .TitleForeColor = QBColor(15)
 .TrailingForeColor = QBColor(2)
 .MonthBackColor = QBColor(8)
End With
Ocx Command cmd(1) = "Change TitleBackColor",
 mvw.Width + 10, 10, 140, 22
Ocx Command cmd(2) = "Change TitleForeColor",
 mvw.Width + 10, 40, 140, 22
Ocx Command cmd(3) = "Change TrailingForeColor",
 mvw.Width + 10, 70, 140, 22
Ocx Command cmd(4) = "Change MonthBackColor",
 mvw.Width + 10, 100, 140, 22
Do : Sleep : Until Me Is Nothing

Sub cmd_Click(Index%)
 Ocx CommDlg cd
 Select Index%
 Case 1 : cd.Color = mvw.TitleBackColor
```

```
  Case 2 : cd.Color = mvw.TitleForeColor
  Case 3 : cd.Color = mvw.TrailingForeColor
  Case 4 : cd.Color = mvw.MonthBackColor
  EndSelect
  cd.ShowColor
  Trace cd.Color
  Select Index%
  Case 1 : mvw.TitleBackColor = cd.Color
  Case 2 : mvw.TitleForeColor = cd.Color
  Case 3 : mvw.TrailingForeColor = cd.Color
  Case 4 : mvw.MonthBackColor = cd.Color
  EndSelect
EndSub
```

See Also

MonthView

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

MonthColumns, MonthRows Properties, ComputeControlSize Method

Purpose

Returns or sets a value that specifies the number of months to be displayed horizontally and vertically. **ComputeControlSize** returns the width and height of a **MonthView** control for a given number of rows and columns.

Syntax

MonthView.**MonthColumns** [= *number*]

MonthView.**MonthRows** [= *number*]

*MonthView.***ComputeControlSize**(*Rows%, Columns%, Width!, Height!*)

Description

The **MonthColumns** and **MonthRows** give you the ability to display more than one month at a time.

The **MonthColumns** property allows you to specify the number of months that will be displayed horizontally. The **MonthRows** property allows you to specify the number of months that will be displayed vertically.

The control can display up to twelve months.

The **ComputeControlSize** method is used to calculate the width and the height of the MonthView control. It takes the *Rows%* and *Columns%* as input parameters and returns the calculated size in the *Width!* and *Height!* variables.

Example

```
OpenW 1, 0, 0
Ocx MonthView mvw
mvw.MonthColumns = 2
mvw.MonthRows = 2
Local Single h, w
~mvw.ComputeControlSize(2, 2, w, h)
// Use of reserved words as below results in an
  error
// Dim Width As Single, Height As Single
// ~mvw.ComputeControlSize(2, 2, Width, Height)
Debug "Width = "; w
Debug "Height = "; h
Win_1.Width = PixelsToTwipX(w + (2 *
  Screen.cxFrame))
Win_1.Height = PixelsToTwipY(h + (2 *
  Screen.cyFrame) + Screen.cyCaption)
Debug.Show
~SetWindowPos(Debug.hWnd, 0, w + 60, 0, 200, 200,
  0)
Do : Sleep : Until Me Is Nothing
Debug.Hide
```

See Also

MonthView

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

ScrollRate Property

Purpose

Returns or sets a value that specifies the number of months that are scrolled when the user clicks one of the scroll buttons in a **MonthView** Ocx.

Syntax

MonthView.**ScrollRate** [= *number*]

Description

The **ScrollRate** property specifies the number of months that are scrolled at once. The **ScrollRate** property allows the user to scroll more than one month at a time.

Example

```
Local Int32 h, w, w1
OpenW 1
Ocx MonthView mvw
~mvw.ComputeControlSize(1, 1, w, h) : w1 =
  TextWidth("Scroll Rate: ")
Ocx Label lbl = "Scroll Rate:", w + 20, 10, w1, 14
Ocx TextBox tb = "", w + w1 + 20, 10, 60, 14 :
  .BorderStyle = 1 ': .ReadOnly = True
Ocx UpDown up : up.BuddyControl = tb : .Increment
  = 1 : .Min = 1 : .Max = 12 : .Value =
  mvw.ScrollRate
Do : Sleep : Until Win_1 Is Nothing

Sub up_Change
```

```
  mvw.ScrollRate = up.Value
EndSub
```

See Also

MonthView

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

ShowToday, ShowWeekNumbers Properties (MonthView)

Purpose

ShowToday returns or sets a value that determines if the current date is displayed at the bottom of the control.

ShowWeekNumbers returns or sets a value that determines if the week numbers are displayed next to each week.

Syntax

MonthView.**ShowToday** [= *boolean*]

MonthView.**ShowWeekNumbers** [= *boolean*]

Description

When the **ShowToday** property is True the current date is displayed.

The **ShowWeekNumbers** property determines whether week numbers are displayed. When False, (Default) week numbers are not displayed.

Week numbers are displayed to the left of the week, and start from the first week of the calendar year.

Example

```
Ocx MonthView mvw : .ShowToday = 0
Ocx CheckBox chk(0) = "Show Today", mvw.Width +
  10, 10, 120, 14
Ocx CheckBox chk(1) = "Show Week Numbers",
  mvw.Width + 10, 27, 120, 14
Do : Sleep : Until Me Is Nothing

Sub chk_Click(Index%)
  mvw.ShowToday = chk(0).Value
  mvw.ShowWeekNumbers = -chk(1).Value
  chk(0).Move mvw.Width + 10
  chk(1).Move mvw.Width + 10
EndSub
```

See Also

MonthView

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

DateClick, DateDblClick, DayClick, SelChange Events

Purpose

Occurs when a date or day on the control is clicked or double clicked.

Syntax

Sub *MonthView*_**DateClick([***index%***],** *DateClicked* **As Date)**

Sub *MonthView*_**DateDblClick([***index%***],** *DateClicked* **As Date)**

Sub *MonthView*_**DayClick([***index%***],** *DayOfWeek%***)**

Sub *MonthView*_**SelChange**(**[***index%***],** *StartDate* **As Date**, *EndDate* **As Date**)

index:An integer that uniquely identifies a form or control if it's in a form or control array.

Description

The **DateClick** event and **DateDblClick** event can be used to respond to the user clicking on a particular date. The *DateClicked* or *DateDblClicked* can be used to determine which date was clicked. **Note:** As at the time of writing (Win8/10), **DateDblClick** does not work; all that happens is that the **DateClick** event is called twice. A workaround has been included in the example below.

DayClick occurs when a day of the week is clicked, which is passed in the parameter *DayOfWeek*.

The **SelChange** event occurs when the user selects a new date or range of dates and has these parameters:

StartDate - The first date in the selection.

EndDate - The last date in the selection.

Example

```
OpenW 1, 0, 0, 242, 200 : Win_1.Caption =
  "Monthview"
Debug.Show
~SetWindowPos(Debug.hWnd, 0, 300, 0, 400, 400, 0)
Ocx MonthView mvw : mvw.MultiSelect = True
Do : Sleep : Until Me Is Nothing
Debug.Hide

Sub mvw_DateClick(DateClicked As Date)
  // Workaround for DateDblClick
  Static tim#
  If Timer - tim# < 0.2 Then
   mvw_DateDblClick(DateClicked) : Exit Sub
  tim# = Timer
  Trace DateClicked
EndSub

Sub mvw_DateDblClick(DateDblClicked As Date)
  Trace DateDblClicked
EndSub

Sub mvw_DayClick(DayOfWeek%)
  Trace DayOfWeek
EndSub
```

```
Sub mvw_SelChange(StartDate As Date, EndDate As
  Date)
  Trace StartDate : Trace EndDate
  Trace mvw.Value
EndSub
```

See Also

MonthView

{Created by Sjouke Hamstra; Last updated: 17/12/2015 by James Gaite}

ClientHeight, ClientWidth, ClientLeft, ClientTop Properties (TabStrip)

Purpose

Return the coordinates of the internal area (display area) of the **TabStrip** control. Read-only

Syntax

object.**ClientHeight**

object.**ClientWidth**

object.**ClientLeft**

object.**ClientTop**

object:TabStrip Ocx

Description

At run time, the client-coordinate properties - **ClientLeft**, **ClientTop**, **ClientHeight**, and **ClientWidth** - automatically store the coordinates of the **TabStrip** control's internal area, which is shared by all **Tab** objects in the control. So that the controls associated with a specific **Tab** appear when that **Tab** object is selected, place the **Tab** object's controls inside a container, such as a **Frame** control, whose size and position match the client-coordinate properties. To associate a container (and its controls) with a **Tab** object, create a control array, such as a **Frame** control array.

All client-coordinate properties use the scale mode of the parent form. To place a **Frame** control so it fits perfectly in the internal area, use the following code:

Example

```
Ocx TabStrip tbs1 = "Tab1", 10, 10, 500, 300
Ocx Frame fr1
fr1.Left = tbs1.ClientLeft
fr1.Top = tbs1.ClientTop
fr1.Width = tbs1.ClientWidth
fr1.Height = tbs1.ClientHeight
tbs1.Add , , "Frame" , , fr1
tbs1.Add , , "No Frame"
Do : Sleep : Until Me Is Nothing
```

Remarks

GFA-BASIC 32 automatically takes care of activating the container after it is associated with the **TabStrip** Ocx.

See Also

TabStrip, Tabs, Tab

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Style, Separators, MultiRow, HotTracking Property (TabStrip)

Purpose

The **Style** property returns or sets the appearance - tabs or buttons - of a **TabStrip** control.

The **Separators** property returns or sets a value that determines whether separators are drawn between buttons on a **TabStrip** control that has the **tabButton** or **tabFlatButton** styles.

The **MultiRow** property returns or sets a value indicating whether a **TabStrip** control can display more than one row of tabs.

HotTracking returns a value that determines whether mouse-sensitive highlighting is enabled.

Syntax

TabStrip.**Style** [= *integer*]

TabStrip.**Separators** [= *boolean*]

TabStrip.**MultiRow** [= *boolean*]

TabStrip.**HotTracking** [= *boolean*]

Description

The **Style** property determines the appearance of the tabs.

tabTabs (0) (Default) Tabs. The tabs appear as notebook tabs, and the internal area has a three-dimensional border around it.

tabButtons (1) Buttons. The tabs appear as regular push buttons, and the internal area has no border around it.

tabFlatButtons (2)Flat buttons. The selected tab appears as pressed into the background. Unselected tabs appear flat.

The **Separators** property specifies if separators are drawn (True). To see the separators, the **TabStrip** control's **Style** property must be set to either **tabButton** or **tabFlatButton**.

The **MultiRow** property specifies whether the control has more than one row of tabs. The number of rows is automatically set by the width and number of the tabs. The number of rows can change if the control is resized, which ensures that the tab wraps to the next row. If **MultiRow** is set to **False**, and the last tab exceeds the width of the control, a horizontal spin control is added at the right end of the **TabStrip** control.

The **HotTracking** property specifies whether hot tracking is enabled or off. Hot tracking is a feature that provides feedback to the user when the mouse pointer passes over the control. With **HotTracking** set to **True**, the control responds to mouse movement by highlighting the header over which the mouse pointer is positioned.

Example

```
Local n As Int32
```

```
Ocx TabStrip tbs = "", 10, 10, 400, 80
For n = 1 To 25 : tbs.Add , , "Tab" & n : Next n
Ocx TabStrip tbt = "", 10, 100, 400, 40
For n = 1 To 25 : tbt.Add , , "Tab" & n : Next n
Ocx TabStrip tbu = "", 10, 160, 400, 100
For n = 1 To 25 : tbu.Add , , "Tab" & n : Next n
' Style property set to the Tabs style.
tbs.Style = tabTabs
' Style property set to the Buttons style:
tbt.Style = tabButtons
tbu.Style = tabFlatButtons
tbu.Separators = True // Only has an effect on
  tabFlatButtons
' Allow more than one row
tbs.MultiRow = True
tbu.MultiRow = True
' Allow hottracking
// Doesn't seem to have an effect on any type
// Hottracking seems automatic on tabTabs and...
// ...non-existant on the rest.
tbs.HotTracking = False
tbt.HotTracking = True
tbu.HotTracking = True
Do : Sleep : Until Me Is Nothing
```

See Also

TabStrip

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Placement, ScrollOpposite Properties (TabStrip)

Purpose

The **Placement** property returns or sets a value that specifies the placement of tabs-top, bottom, left, or right.

ScrollOpposite returns or sets a boolean that determines how remaining rows of tabs in front of a selected tab are repositioned.

Syntax

TabStrip.**Placement** [= *integer*]

TabStrip. **ScrollOpposite** [= *boolean*]

Description

The **Placement** property specifies the tabs' location:

0 or 1 - (Default) The tabs appear at the top of the control.

2 - The tabs appears at the bottom of the control.

3 - The tabs appears at the control's left.

4 - The tabs appears at the control's right.

The **ScrollOpposite** property specifies how the remaining tabs will be repositioned. When False (default), the remaining tabs remain on the same side of the control.

When True, The row of tabs in front of the selected tab are repositioned at the opposite side of the control.

ScrollOpposite only works correctly with **Style** = **tabTabs**, not with buttons.

Example

```
OpenW Fixed 1
Global Int32 n, h, w
h = TwipsToPixelY(Win_1.Height) - (2 *
  Screen.cyFixedFrame) - (Screen.cyCaption)
w = TwipsToPixelX(Win_1.Width) - (2 *
  Screen.cxFixedFrame) + 1
Ocx TabStrip tbs
tbs.Style = tabTabs
tbs.ScrollOpposite = False
cmd_Click(0)
For n = 1 To 10 : tbs.Add , , "Tab " & n : Next n
Ocx Command cmd(1) = "Top", (w - 60) / 2, (h / 2)
  - 40, 60, 22
Ocx Command cmd(2) = "Bottom", (w - 60) / 2, (h /
  2) + 18, 60, 22
Ocx Command cmd(3) = "Left", (w / 2) - 80, (h -
  22) / 2, 60, 22
Ocx Command cmd(4) = "Right", (w / 2) + 20, (h -
  22) / 2, 60, 22
Ocx CheckBox chk = "ScrollOpposite", (w - 100) /
  2, (h / 2) + 60, 100, 14 : chk.BackColor =
  RGB(255, 255, 255)
Do : Sleep : Until Me Is Nothing

Sub chk_Click
  tbs.ScrollOpposite = 1 - chk.Value
EndSub

Sub cmd_Click(Index%)
```

```
  Select Index%
  Case 0, 1 : tbs.Move 0, 0, w, 40
  Case 2 : tbs.Move 0, h - 40, w, 40
  Case 3 : tbs.Move 0, 0, 40, h
  Case 4 : tbs.Move w - 40, 0, 40, h
  EndSelect
  tbs.Placement = Index%
EndSub
```

Known Issues

ScrollOpposite doesn't seem to work at all,

See Also

TabStrip

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

TabFixedHeight, TabFixedWidth, TabWidthStyle, TabMinWidth Properties (TabStrip)

Purpose

Return or set the fixed height and width of all **Tab** objects in a **TabStrip** control, but *only* if the **TabWidthStyle** property is set to **tabFixed**.

TabWidthStyle returns or sets a value that determines the justification or width of all **Tab** objects in a **TabStrip** control.

The **TabMinWidth** property returns or sets the minimum allowable width of a tab.

Syntax

TabStrip.**TabFixedHeight** [= *integer*]

TabStrip.**TabFixedWidth** [= *integer*]

TabStrip.**TabWidthStyle** [= *integer*]

TabStrip.**TabMinWidth** [= *integer*]

Description

The **TabFixedHeight** and **TabFixedWidth** properties specify the number of pixels or twips of the height or width of a **TabStrip** control. The scale used for *integer* is dependent on the **ScaleMode** of the container.

The **TabFixedHeight** property applies to all **Tab** objects in the **TabStrip** control. It defaults either to the height of the font as specified in the **Font** property, or the height of the **ListImage** object specified by the **Image** property, whichever is higher, plus a few extra pixels as a border. If the **TabWidthStyle** property is set to **tabFixed**, and the value of the **TabFixedWidth** property is set, the width of each **Tab** object remains the same whether you add or delete **Tab** objects in the control.

The **TabWidthStyle** property determines whether tabs are justified or set to a fixed width, it can take the following values:

tabJustified (0)	(Default) Justified. If the **MultiRow** property is set to **True**, each tab is wide enough to accommodate its contents and, if needed, the width of each tab is increased so that each row of tabs spans the width of the control. If the **MultiRow** property is set to False, or if there is only a single row of tabs, this setting has no effect.
tabNonJustified (1)	Non-justified. Each tab is just wide enough to accommodate its contents. The rows are not justified, so multiple rows of tabs are jagged.
tabFixed (2)	Fixed. All tabs have an identical width, which is determined by the **TabFixedWidth** property.

At design time, you can set the **TabWidthStyle** property on the Properties Page of the **TabStrip** control. The setting of the **TabWidthStyle** property affects how wide each **Tab** object appears at run time.

The **TabMinWidth** property specifies the minimum width of a **Tab** object. The scale used for *number* is determined by the **ScaleMode** property of the container. The **TabMinWidth** property has no effect if the **TabWidthStyle** property is set to **tabFixed.**

Example

```
Ocx TabStrip ts(1) = "", 10, 10, 700, 110 :
  ts(1).MultiRow = True : ts(1).TabWidthStyle =
  tabJustified
Ocx TabStrip ts(2) = "", 10, 140, 700, 110 :
  ts(2).MultiRow = True : ts(2).TabWidthStyle =
  tabNonJustified
Ocx TabStrip ts(3) = "", 10, 270, 700, 40 :
  ts(3).TabWidthStyle = tabFixed :
  ts(3).TabFixedWidth = 90
Ocx TabStrip ts(4) = "", 10, 330, 700, 40 :
  ts(4).TabWidthStyle = tabFixed :
  ts(4).TabFixedWidth = 120 : ts(4).TabFixedHeight
  = 35
Ocx TabStrip ts(5) = "", 10, 390, 700, 40 :
  ts(5).TabMinWidth = 50
Local Int32 m, n
For m = 1 To 5
  For n = 1 To 40
    If Odd(n) Then ts(m).Add , , "Tab" & n
    If Even(n) Then ts(m).Add , , "Tab" & n & "
      (Even)"
  Next n
Next m
Do : Sleep : Until Me Is Nothing
```

See Also

TabStrip

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

Add, AddItem Method (TabStrip, Tabs)

Purpose

Adds a **Tab** to a **Tabs** collection in a **TreeView** control and returns a reference to the newly created **Tab** object.

Syntax

*TabStrip***.Add[Item]**(index, key, text, image, ocx)

*Tabs***.Add**(index, key, text, image, ocx)

index, key, text, image, ocx: Variant exp

Description

The **TreeView** Ocx has the **AddItem** and **Add** methods, which act exactly the same. The **Tabs** object supports the **Add** method only.

index	An integer specifying the position where you want to insert the **Tab**. If you don't specify an index, the **Tab** is added to the end of the **Tabs** collection.
key	Optional. A unique string that can be used to retrieve the **Tab** with the **Item** method.
text	Optional. The string that appears on the **Tab**. This is equivalent to setting the **Caption** property of the new **Tab** object after the object has been added to the **Tabs** collection.
image	Optional. The index of an image in an associated

	ImageList control. This is equivalent to setting the **Image** property of the new **Tab** object after the object has been added to the **Tabs** collection.
ocx	Optional. The container Ocx control to display in the client area of the **TabStrip**.

Use the **Key** property to reference a member of the **Tabs** collection if you expect the value of an object's **Index** property to change, such as by dynamically adding objects to or removing objects from the collection. The **Tabs** collection is a 1-based collection.

As a **Tab** object is added it is assigned an index number, which is stored in the **Tab** object's **Index** property. This value of the newest member is the value of the **Tab** collection's **Count** property.

Because the **Add** method returns a reference to the newly created **Tab** object, it is most convenient to set properties of the new **Tab** using this reference.

Example

```
Form Hidden Center frm1 = "TabStrip", , , 400, 300
Ocx TabStrip tbs = , 20, 20, ScaleWidth - 40,
  ScaleHeight - 40
Ocx Frame fr1 = "Tab #1"
Ocx Frame fr2 = "Tab #2"
Ocx Frame fr3 = "Tab #3"
Ocx Frame fr4 = "Tab #4"
Dim tab As Tab
tbs.Tabs.Add 1, , fr1.Caption , , fr1
tbs.AddItem 2, , fr2.Caption, , fr2
tbs.Add 3, , fr3.Caption, , fr3
Set tab = tbs.AddItem(4, , , , fr4)
tab.Caption = fr4.Caption
```

```
frm1.Show
tbs(2).Selected = True
Do
  Sleep
Until Me Is Nothing
```

This example shows just one way to add **Frame** OCXs as containers to the **Tabs** collection.

Remarks

Note - There are several ways to add containers to a **TabStrip** Ocx control. See the description of the **TabStrip** control.

GFA-BASIC 32 specific

Instead of explicitly using the **Tabs** collection to access a **Tab** element, you can use a shorter notation. First, the **TabStrip** Ocx supports an **Item** property:

tbs.**Item**(idx)tbs.**Tabs**.**Item**(idx)

Like the **Item** method of tbs.**Tabs, Item** is the default method of **TabStrip**. Therefore, a **Tab** object can be accessed as follows:

tbs(idx)tbs.**Tabs**(idx)

tbs!idxtbs.**Tabs**!idx

Each dot saves about 30 bytes of code.

To enumerate over the **Tabs** collection of a **TabStrip** Ocx, use **For Each** on the Ocx control directly, like:

```
Local tab1 As Tab
```

```
For Each tab1 In tbs
  DoSomething(tab1)
Next
```

See Also

TabStrip, Tab, Tabs

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

NextTab, PrevTab Methods (TabStrip)

Purpose

Activates the next or previous **Tab** in a **TabStrip** control.

Syntax

TabStrip.**NextTab**

TabStrip.**PrevTab**

Description

NextTab activates the next **Tab** in **TabStrip** control. **PrevTab** selects the previous **Tab**. Both methods are circular. A Change event is generated.

Example

```
Local n As Int32
OpenW Fixed 1
Ocx TabStrip tbs = "", 0, 10,
  TwipsToPixelX(Win_1.Width), 40
tbs.TabFixedWidth = TwipsToPixelX(Win_1.Width) /
  11 : tbs.TabWidthStyle = tabFixed
For n = 1 To 10 : tbs.Add , , "Tab " & n : Next n
Ocx Command cmd(1) = "<<< Previous Tab", 10, 70,
  120, 22
Ocx Command cmd(2) = "Next Tab >>>", 150, 70, 120,
  22
Do : Sleep  : Until Me Is Nothing
```

```
Sub cmd_Click(Index%)
  Select Index%
  Case 1 : tbs.PrevTab
  Case 2 : tbs.NextTab
  EndSelect
EndSub
```

See Also

TabStrip, Change

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Change, BeforeChange Event (TabStrip)

Purpose

The **Change** event occurs when the currently selected tab has changed. The **BeforeChange** event occurs when the currently selected tab is about to change.

Syntax

Sub TabStrip_**Change**

Sub TabStrip_**BeforeChange**(Cancel?)

Description

BeforeChange is generated before **Change** and can be used to cancel the tab change by setting *Cancel*? to **True**. Inside **BeforeChange** the **SelectedItem** property (or SelectedIndex) still specifies the current tab.

When a new tab is selected **Change** is invoked. Use **SelectedItem** or **SelectedIndex** to obtain the current **Tab** object.

Example

```
Ocx TabStrip tbs = "", 0, 0, 220, 25
Ocx Option opt(1) = "Option Box 1", 10, 40, 100,
  14 : tbs.Add 1, , "Option 1"
Ocx Option opt(2) = "Option Box 2", 10, 60, 100,
  14 : tbs.Add 2, , "Option 2"
```

```
Ocx Option opt(3) = "Option Box 3", 10, 80, 100,
  14 : tbs.Add 3, , "Option 3"
Ocx Option opt(4) = "Option Box 4", 10, 100, 100,
  14 : tbs.Add 4, , "Option 4"
opt(1).Value = 1
Do : Sleep : Until Me Is Nothing

Sub tbs_Change
  opt(tbs.SelectedIndex).Value = 1
End Sub

Sub tbs_BeforeChange(Cancel?)
  If MsgBox("Tab change allowed?", MB_OKCANCEL) =
    IDCANCEL
    Cancel? = True
  EndIf
EndSub
```

See Also

TabStrip, Tabs, Tab

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

AutoPlay, Center, Transparent Properties (Animation)

Purpose

Define the window styles used with the Animation object.

Syntax

Animation.**AutoPlay** [= *boolean*]

Animation.**Center** [= *boolean*]

Animation.**Transparent** [= *boolean*]

Description

AutoPlay starts playing the animation as soon as the AVI clip is opened.

Center centers the animation in the animation control's window.

Transparent draws the animation using a transparent background rather than the background color specified in the animation clip.

Example

See Also

Animation

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Close, Open, Play, Seek, Stop Methods (Animation)

Purpose

Methods to control the **Animation** Ocx.

Syntax

Animation.**Close**

Animation.**Open** file$

Animation.**Play** [*repeat, start, end*]

Animation.**Seek**

Animation.**Stop** *frame%*

repeat, start, end:Variant

Description

The **Close** method causes the **Animation** control to close the currently open AVI file. If there was no file loaded, **Close** does nothing, and no error is generated.

Open opens an .avi file to play. If the **AutoPlay** property is set to **True**, then the clip will start playing as soon as it is loaded. It will continue to repeat until the .avi file is closed or the **Autoplay** property is set to **False**.

Play [*repeat, start, end*] plays an .avi file.

*repeat*Optional. Integer that specifies the number of times the clip will be repeated. The default is -1, which causes the clip to repeat indefinitely.

*start*Optional. Integer that specifies the starting frame. The default value is 0, which starts the clip on the first frame. The maximum value is 65535.

*end*Optional. Integer that specifies the ending frame. The default value is -1, which indicates the last frame of the clip. The maximum value is 65535.

Seek directs an animation control to display a particular frame of an AVI clip. The control displays the clip in the background while the thread continues executing. *frame%* is a zero-based index of the frame to display.

Stop stops the play of an .avi file in the **Animation** control. The **Stop** method stops only an animation that was started with the **Play** method. Attempting to use the **Stop** method when the **Autoplay** property is set to True returns an error

Example

Remarks

To stop a file from playing, use the **Stop** method. However, if the **Autoplay** property is set to **True**, set **Autoplay** to **False** to stop the file from playing.

See Also

Animation

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Start, Stop Events (Animation)

Purpose

Occur when an **Animation** control has started or stopped playing.

Syntax

Sub *Animation*_**Start**

Sub *Animation*_**Stop**

Description

Start is generated when the associated AVI clip has started playing.

Stop is generated when the associated AVI clip has stopped playing.

Example

```
Sub ani1_Start
  Me.Caption = "Start"
  Trace "Start"
End Sub

Sub ani1_Stop
  Me.Caption = "Stop"
  Trace "Stop"
End Sub
```

See Also

Animation

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

BuddyControl, LeftAlign, and Horizontal Properties (UpDown)

Purpose

Sets or returns information determining the connection with a buddy control.

Syntax

UpDown.**BuddyControl** [= Ocx]

UpDown.**LeftAlign** [= Boolean]

UpDown.**Horizontal** [= Boolean]

Description

An **UpDown** control has a pair of arrow buttons which the user can click to increment or decrement a value, such as a scroll position or a value in an associated control, known as a buddy control.

To the user, an **UpDown** control and its buddy control often look like a single control. The buddy control can be any control that can be linked to the **UpDown** control through the **BuddyControl** property, and usually displays data, such as a **TextBox** control or a **Command** control.

The **UpDown** control can be positioned to the right (default) or left of its buddy control with the **LeftAlign** property. The **BuddyControl** property sets or returns the

Ocx control used as the buddy control. The arrows may be positioned vertically (default) or horizontally with the **Horizontal** property.

Example

```
Ocx UpDown up = "", 99, 10, 15, 18
Ocx TextBox tb = "0", 70, 11, 40, 16 :
  tb.BorderStyle = 1
up.LeftAlign = True // Moves it to the left of the
  Textbox
up.Horizontal = True // Converts UpDown to virtual
  Left/Right
up.BuddyControl = tb // Combines the UpDown OCX
  with the Textbox
Do : Sleep : Until Me Is Nothing
```

Remarks

Changing anyone of the these properties has an effect on the assigned Buddy Control, usually foreshortening it. To see the effect, move the **BuddyControl** line up two and you will notice that the textbox all but disappears.

See Also

UpDown

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Format Method (UpDown)

Purpose

Sets or gets a format string specifying the format instructions for the numeric value for the buddy control.

Syntax

UpDown.**Format =** [*format*]

format:sexp

Description

The **Format** property specifies the string to display in the buddy control. The text for the buddy control is not set with the **Text** or **Caption** property of the buddy control itself.

The **Format** property uses the same format string as is used with the **Format** function. The type of data in the buddy control is not limited to integer values only. The **Format** property can specify a **Date** format as well.

Example

```
Form frm1 = "UpDown", , , 200, 200
Ocx TextBox tbu = "??", 5, 5, 150, 24
.Appearance = 1
Ocx UpDown updn
updn.BuddyControl = tbu
updn.Max = #12/31/2999#
updn.Value = Date
updn.Format = "Long Date"
```

```
Do
  Sleep
Until Me Is Nothing
```

Remarks

The text "??" of the TextBox OCX is not displayed.

The **Value** for the UpDown OCX is set to the current date.

The text displayed in the TextBox is formated using the format specification in the **Format** property of the UpDown OCX.

See Also

UpDown, Format()

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Value, Increment, Max, Min, Wrap Property (UpDown)

Purpose

Returns or sets the value of an object.

Syntax

UpDown.**Value** [= *Double*]

UpDown.**Min** [= *Double*]

UpDown.**Max** [= *Double*]

UpDown.**Increment** [= *Double*]

UpDown.**Wrap** [= *boolean*]

Description

The **Value** property sets or returns the current position of the scroll value. The **Value** property specifies the current value within the range of the **Min** and **Max** properties (default is 0). This property is incremented or decremented when the arrow buttons are clicked.

The **Min** and **Max** properties sets or returns the maximum value of the scroll range for the **UpDown** control. The default value for **Min** is 0 and for **Max** is 100. The settings of the **Min** and **Max** properties determine whether the value is incremented or decremented when the arrow buttons are clicked. If the **Max** property is less than the **Min** property, the **UpDown** control operates in the reverse direction.

Pressing the up or right arrow always causes the **Value** property to approach the **Max** value. Pressing the down or left arrow always causes the Value property to approach the **Min** value.

The **Wrap** property sets or returns a value that determines whether the control's **Value** property wraps around to the beginning or end once it reaches the **Max** or **Min** value.

The **Increment** property determines the amount the **Value** property changes when you click the arrow buttons on the **UpDown** control. The default value is 1. Clicking the up or right arrow, causes **Value** to approach the **Max** property by the amount specified by the **Increment** property. Clicking the down or left arrow, causes **Value** to approach the **Min** property by the amount specified by the **Increment** property.

Example

```
Ocx TextBox tb = "", 10, 10, 45, 15 : .BorderStyle
  = 1 : .ReadOnly = True
Ocx UpDown up : .BuddyControl = tb  : .Min = -3 :
  .Max = 10 : .Increment = 3 : .Wrap = True :
  .Value = 4
Ocx Label lbl = "up.Value = 4", 10, 40, 100, 14
Do : Sleep : Until Me Is Nothing

Sub up_Change
  lbl.Text = "up.Value = " & up.Value
EndSub
```

See Also

UpDown

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

UpClick, DownClick, and Change events (UpDown)

Purpose

This events occurs when the down or left arrow button is clicked.

Syntax

Sub *UpDown*_**UpClick**()

Sub *UpDown*_**DownClick**()

Sub *UpDown*_**Change**()

Description

Using the **UpClick** and **DownClick** events, you can control exactly how the **UpDown** control scrolls through a series of values.

The **Change** event occurs whenever the **Value** property changes. The **Value** property can change through code, by clicking the arrow buttons, or by changing the value in a buddy control when the **BuddyControl** property is set.

For example, if you want to allow the end user to scroll rapidly upward through the values, but slower going down through the values, you can set reset the **Increment** property to different values, as shown below:

Example

```
Ocx TextBox tb = "", 10, 10, 50, 15 : .BorderStyle
  = 1 : .ReadOnly = True
Ocx UpDown upd : .BuddyControl = tb : .Max = 100 :
  .Value = 10
Ocx Label upl = "<= upd", 65, 10, 120, 15
Ocx TextBox tb1 = "", 10, 30, 50, 15 :
  .BorderStyle = 1 : .ReadOnly = True
Ocx UpDown up1 : .BuddyControl = tb1 : .Max = 5 :
  .Value = 1
Ocx Label up1l = "Increment Value of upd", 65, 30,
  120, 15
Ocx Label lbl = "", 10, 50, 140, 15
Do : Sleep : Until Me Is Nothing

Sub upd_UpClick
  lbl.Caption = "upd Up Button Clicked"
EndSub

Sub upd_DownClick
  lbl.Caption = "upd Down Button Clicked"
EndSub

Sub up1_Change
  upd.Increment = up1.Value
EndSub
```

See Also

UpDown

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Gfa_Type and Gfa_Types

Syntax

Dim type As **Gfa_Type**

Dim types As **Gfa_Types**

Description

A **Gfa_Type** item contains the properties that allow you to get information about a user-defined type like its name, elements, and their type.

Property	**Description**
Count	Returns the number of elements of the type.
Name(*n*)	The name of the element n in the user-defined type. Name(0) returns the name of the Type itself.
Index(*e$*)	Returns the index of element e$.
Size(*n*)	Returns the memory size of element n in bytes.
Type(*n*)	A value indicating the type of element n (basInt, basFixedString, etc)
TypeName(*n*)	A string describing the type of element n (Integer, String)
Offset(*n*)	Returns the offset from the start of element n in bytes
BitSize(*n*)	Returns the memory size element n in bits
BitOffset(*n*)	Returns the offset of element n from the

	beginning of the type in bits.
TypeObj*(n)*	Returns a **Gfa_Type** object for the element when Type(n) > 65535.
IsArray*(n)*	Returns True when the element is an array. False for a Variant containing an array.
LBound*(n)*	Returns the smallest available subscript for the specified dimension of the array
UBound*(n)*	Returns the largest available subscript for the specified dimension of the array
ArrSize*(n)*	The allocated memory for the array in bytes.

Obtaining type information is important when you develop a custom debugger. However, the type information can be obtained at design time as well. The following example shows all user defined types currently in use in your program. If you perform a syntax check before pressing App + T, the Gfa_Types collection provides the type elements as well.

Gfa_Types is a collection containing **Gfa_Type** items. A **Gfa_Type** item contains information about a user-defined type.

The **Gfa_Types** collection allows you to enumerate over all types defined in the program.

```
Dim t As Gfa_Type
For Each t In Gfa_Types
  Debug t.Name(0)
Next
```

Example

Enumerate all Types in a program.

```
Sub Gfa_App_T
  Debug.Show
  Dim udt As Gfa_Type, i%
  For Each udt In Gfa_Types
    Try
      Debug "Type ";udt.Name(0)
      For i = 1 To udt.Count
        Debug "  ";udt.Name(i);
        If udt.IsArray(i) Then
          Debug "
            (";udt.LBound(i);"..";udt.UBound(i);")";
        EndIf
        Debug " As ";udt.TypeName(i);#9;" // Type:
          ";udt.Type(i)
      Next
      Debug "EndType"
    Catch
      Debug "Error in Print_type ";Err;Err$
    EndCatch
  Next
  Debug "- Did you perform a syntax check first?"
  ' Sample
  Type test
    tstl(6 .. 7) As Long
    tsts As String*20
    rc(2) As RECT
  EndType
  Type RECT
    - Int32 left, top, right, bottom
  EndType
EndSub
```

Remarks

According to German documentation the .Type property returns a variant-type constant. When .Type = VT_I4 the variable is an integer and when .Type is VT_I4 | VT_BYREF a ByRef Integer Parameter.
VT_I4 | VT_ARRAY is a global or static Integer array, VT_I4 | VT_BYREF | VT_ARRAY a local Integer array or a ByRef Integer array parameter.
The same is true for VT_I2 (Short) VT_UI2 (Card), VT_UI1 (Byte), VT_BOOL (Boolean), VT_R4 (Single), VT_R8 (Double), VT_DATE (Date), VT_VARIANT (Variant), VT_CY (Currency) and VT_I8 (Large). With user-defined types the property is a value greater than 4 billion, with ByRef a user-defined type parameter is odd, otherwise even.

See Also

Gfa_Var, Gfa_Vars

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_Key Function

Syntax

shortcut$ = **Gfa_Key**(x)

x: as predefined constant representing a sub

Description

Returns the assigned shortcut for the given keyboard event sub (Func) in the current GLL. This is an informational function only that provides current keyboard assignments of the subs in the GLL where this function is used. **Gfa_Key** does not provide information of the keyboard assignments of other currently loaded GLLs.

The parameter *x* is not a variable and it has no type; it is simply a strange GFA-BASIC 32 constant that identifies a keyboard sub event. For instance, the parameter *sc+A* represents the sub Gfa_Ex_A, and sc+F11 specifies the event sub Gfa_SCF11. Valid values are *sc+0*, *App+A*, *App+s+I*, *SCF11*, etc. The constant is best looked up in the Key Assignment dialog box. The parameter is not a string, but the return value is!

Example

```
Dim ActiveKey$ = Gfa_Key(sc+A)
If Len(ActiveKey$)
  MsgBox "The Gfa_Ex_A sub is executed with the
    keyboard shortcut " & ActiveKey$
Else
```

```
  MsgBox "There is no Gfa_Ex_A sub in this GLL, or"
    #10 _
    "Gfa_Ex_A has been disabled because it's
      shortcut has been removed, or" #10 _
    "it has been disabled because another GLL uses
      the keyboard shortcut."
EndIf
```

Remarks

If the return value is an empty string, then

1. There is no event sub with that name, or

2. There is no key assigned to the Gfa_ event sub, or

3. The key assignment is disabled, because another GLL, loaded earlier, uses the keyboard shortcut.

See Also

Gfa_Ex_

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_Menu Commands and Functions

The GFA-BASIC 32 editor extension functions can be executed through the use of a set of predefined keystrokes or by selecting a menu item from the Extra submenu. To connect some functionality to a keyboard shortcut the name of the procedure should conform to a special format that describes the shortcut to use (Gfa_Ex_, Gfa_App_, etc).

Another way of invoking an extensibility function is to select a menu item from the Extra menu. To provide a menu entry the GLL must add a menu entry to the Extra menu in **Gfa_Init** sub using **Gfa_AddMenu**. This function adds an item to the menu and specifies a menu event sub that handles the menu entry when it is selected. Other commands are available to modify an appended menu entry. An item can be enabled or disabled, gets a checkmark in front of it, or its text can be changed afterwards.

Syntax

```
[id = ]Gfa_AddMenu("Entry", eventsub)
f = Gfa_MenuCheck(id)          (id As Int, f? as Boolean)
Gfa_MenuCheck(id) = f          (id As Int, f? As Boolean)
d = Gfa_MenuDesc(id)           (id As Int, d As String)
Gfa_MenuDesc(id) = d           (id As Int, d As String)
f = Gfa_MenuEnable(id)         (id As Int, f As Boolean)
Gfa_MenuEnable(id) = f         (id As Int, f As Boolean)
t = Gfa_MenuText(id)           (id As Int, t$ as String)
Gfa_MenuText(id) = t           (id As Int, t$ as String)
```

Description

Gfa_AddMenu this appends the menu item "Entry" to the Extra-Menu of the GFA-BASIC 32 IDE. The text may contain an ampersand (&) to provide keyboard shortcut. The **Gfa_AddMenu** is usually invoked in the **Gfa_Init** sub.

Gfa_MenuCheck(id) [=] returns or sets a value that determines whether a check mark is displayed next to a menu item.

Gfa_MenuDesc(id) [=] returns or sets the descriptive text displayed in the status bar for a menu item. The text is limited to 159 bytes.

Gfa_MenuEnable(id) [=] returns or sets a value that determines whether the specified menu item can respond to user-generated events.

Gfa_MenuText(id) [=] returns or sets the text displayed for a menu item. The text is limited to 159 bytes.

Example

```
Sub Gfa_Init
  Global IdxCreateCode%           ' Declare Globals
    in Gfa_Init
  IdxCreateCode = Gfa_AddMenu("&Create
    code"#9"Ctrl+Shift+C", MenuCreateCode)
  Gfa_MenuDesc(IdxCreateCode) = "Inserts a code
    snippet at current cursor position"
EndSub

Sub MenuCreateCode(i%)             ' a ByRef integer
  parameter
  Debug "Menu selection ID "; i%
```

```
End Sub
```

Remarks

The Sub *MenuCreateCode* is invoked when the menu entry is selected. The event sub must contain one Integer parameter (not **ByVal**). The integer parameter is a value between 1 and 50 (the maximum number of entries) representing its "position" in the Extra submenu. The numbers are given to the entries automatically when they are appended using **Gfa_AddMenu**. This value is returned when the function version of **Gfa_AddMenu** is used instead. Keeping this value in a global variable is necessary when the menu entry is later manipulated with **Gfa_MenuDesc**, **Gfa_MenuEnable**, or **Gfa_MenuText**.

Another use of saving the menu ID value is to use it later in a general menu event subroutine to differentiate between the menu items in a Switch statement. However, this is not encouraged because the execution of a menu event sub for one entry is faster.

The Win API menu ID is calculated by adding 2499 to the value returned by Gfa_AddMenu.

Note Multiple editor extensions can add menu entries to the Extra menu. Later, when a GLL is unloaded, their entries are unloaded as well. This will not lead to renumbering the ID values of the menu entries that are currently in the Extra menu.

See Also

Gfa_Ex_, Gfa_Init

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_Goto Command

Syntax

Gfa_Goto

Description

Displays the GoTo dialog box with the current line as default value. Relative jumps are possible by specifying the number of lines to jump. A positive value will jump forwards a negative value backwards.

See Also

Gfa_TopLine, Gfa_Line

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Gfa_Undo

Syntax

Gfa_Undo

Description

Gfa_Undo reverses previous editing actions on the text. The undo stack can hold only 64 actions.

See Also

Gfa_Copy

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Gfa_CommentBlock, Gfa_UnCommentBlock Commands

Syntax

Gfa_CommentBlock

Gfa_UnCommentBlock

Description

Gfa_CommentBlock comments of a block of selected lines with the ° comment mark to differentiate the comments from a normal commented line. After commenting the line indention is removed.

Gfa_UnCommentBlock removes °comments of a block of selected lines. Only lines starting with °comments are affected, other lines are not processed. The IDE does not define keyboard shortcuts.

Example

```
// Define keyboard shortcuts for block commenting

Sub Gfa_Ex_C    // Shift+Ctrl+C - CommentBlock
 If GfaLine <> Gfa_SelLine
  Gfa_CommentBlock
 EndIf
EndSub

Sub Gfa_Ex_U    // Shift+Ctrl+U - UnCommentBlock
```

```
  Gfa_UncommentBlock
EndSub
```

See Also

Gfa_SelLine

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Gfa_Insert, Gfa_Replace Commands

Syntax

Gfa_Insert s$

Gfa_Replace s$

Description

Gfa_Insert inserts a string at the current position as if it is pasted from the clipboard. In case of selected text, the selection is replaced.

The string may contain end of line markers (#10 (Line Feed)). A Carriage Return (#13) is ignored, so that MS-DOS typical end of line markers (#13#10) can be used.

The insertion does not take the current overwrite modus into account, **Gfa_Insert** always inserts the text. To do a destructive insertion, either select the text to destroy before using **Gfa_Insert**, or use **Gfa_Replace**. A disadvantage of **Gfa_Replace** is its limitation in that it doesn’t replace across line boundaries.

Gfa_Replace - The contents of the string s$ replaces Len(s$) characters in the text of the current line starting at **Gfa_Col**. When the line contains a selection, the selection is replaced.

Line boundaries can not be crossed, replacement is stopped at the first LF (or any ASCII code < 32). So, the replacement string should not contain control characters, use **Gfa_Insert** instead.

Gfa_Replace s$ is equivalent to

```
Txt$ = Gfa_Text
Mid(Txt$, Gfa_Col) = s$
Gfa_Text = Txt$
```

Bug: When the replacement string overwrites the length of the current line, random characters are added.

Remarks

This command has nothing to do with the Replace menu item in the Edit menu.

See Also

Gfa_Text, Gfa_DeleteLines, Gfa_InsertLines

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_DeleteLines, Gfa_InsertLines Command

Syntax

Gfa_DeleteLines [n = 1]

Gfa_InsertLines [n = 1]

Description

Gfa_DeleteLines [n = 1] deletes one or more lines. When n is specified then, for n>0 the current line (**Gfa_Line**) and n - 1 following lines will be deleted.

Gfa_InsertLines [n = 1] inserts one or more empty lines above the current line (**Gfa_Line**). When n is used, then for n>0 n empty lines are inserted. **Gfa_InsertLines** 1 is equivalent to **Gfa_CtrlN**.

See Also

Gfa_Insert, Gfa_Replace, Gfa_Text, Gfa_CtrlN

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Gfa_Error, Gfa_NextError, Gfa_PrevError

Syntax

syntaxerr? = **Gfa_Error**

Gfa_Error = syntaxerr?

Gfa_NextError

Gfa_PrevError

Description

Gfa_Error returns True when the current line (**Gfa_Line**) contains (syntax) errors.

Using Gfa_Error= True marks the current line as having a syntax error and the line can then be left without letting GFA-BASIC 32 parse the line (**Gfa_Update**). The line is then displayed in the syntax-error color.

When changing an erroneous line to False, so when **Gfa_Error** for the current line is True, the line is marked as **Gfa_Changed** = True. Consequently, the line is parsed (**Gfa_Update**) when the cursor leaves the line.

Gfa_NextError jumps to the next line marked as erroneous.

Gfa_PrevError jumps to the previous line marked as erroneous.

Example

Remarks

See Gfa_Changed for an explanation on syntax checking.

See Also

Gfa_Changed, Gfa_Update

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Gfa_DebMenu, Gfa_DebOn

Syntax

Gfa_DebMenu [= menutext$]

Gfa_DebOn n%, procname   (n in 0 … 4, procname as name of sub)

Description

Appends a new menu item to the tray icon popup menu. This is the popup menu that is displayed when the debug tray icon is activated with a right button click. To set an event sub for the new entry use a special version of **Gfa_DebOn**.

```
Sub Gfa_OnRun
  Gfa_DebMenu = "Show local variables"
  Gfa_DebOn 0, My_DebMenu          ' Defines the event
    sub for the menu item
EndSub
```

Note The **Gfa_DebOn** method is otherwise used to set event subs for clicking on the tray icon.

Gfa_DebOn *n%, procname* set event subs for mouse clicks on the debug tray icon. This command is usually invoked in the **Gfa_OnRun** event sub, that is, this procedure offers the first opportunity to set debugging events and variables.

This command allows you to setup an additional 5 event subs to help you in creating a custom debugger. The first event sub you might add is a sub to respond to the selection of the new menu item from the **Gfa_DebMenu** method. To do so, use n% = 0 and specify the name of the sub to call in case the menu item is selected.

The other events you can respond to are the left and middle button click events. For n% = 1 you can define the event sub for a click with the left mouse button, for n% = 2 a left double click.

When n% = 3 and n% = 4 you can do the same for the middle mouse button (if available).

Example

```
' Left Button clicks
Gfa_DebOn 1, Deb_LClick          ' pick a name
Gfa_DebOn 2, Deb_LDblClick       ' what's in a
 name?
' Middle Button clicks
Gfa_DebOn 3, Deb_MClick          '
Gfa_DebOn 4, Deb_MDblClick       ' middle button
 double click on the tray icon.

Sub Deb_LClick
  MsgBox "Left button click on the debug tray
    icon!"
  Gfa_DbStep
EndSub

Sub Deb_LDblClick
  MsgBox "Left button double click on the debug
    tray icon!"
  Gfa_DbStep
EndSub

Sub Deb_MClick
  MsgBox " Middle button click on the tray icon."
  Gfa_DbStep
EndSub

Sub Deb_MDblClick
  MsgBox " Middle button double click on the tray
    icon."
  Gfa_DbStep
EndSub
```

See Also

Gfa_DbStep, Gfa_DbOn, Gfa_DbOff

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_Vars Collection

Syntax

Dim vs As **Gfa_Vars**

Set vs = **Gfa_Vars**(subcol$)

Description

The **Gfa_Vars** collection object represents all or part of the variables used in an application. A **Gfa_Vars** collection consists of **Gfa_Var** items. A **Gfa_Var** item contains the properties that allow you to get information about the variable like its name, type, location, and value.

The only way to obtain a **Gfa_Vars** collection object, is by invoking the function **Gfa_Vars(***subcol*$**)** that returns a **Gfa_Vars** collection as specified in the function's parameter value. For instance

```
Dim globalvars As Gfa_Vars
Set globalvars = Gfa_Vars("")    ' the global
 variables
```

The parameter designates the sub collection of variables. The parameter is of type Variant and can be one of the following values:

Parameter	**Returns a Gfa_Vars collection object with**	**Alias**
""	All global variables..	**Gfa_Globals**
"-"	All local and static variables of the main	

	program.	
"Procname"	All local and static variables of the specified procedure. The name is case sensitive!	**Gfa_Vars**!Procname
"1" or 1	All local and static variables of the current procedure. Available in Gfa_Tron and Gfa_TronBook proc only.	**Gfa_Vars1 or Gfa_Vars!1**
"2" or 2	All local and static variables *of the caller* of the current procedure. Available in Gfa_Tron and Gfa_TronBook proc only.	**Gfa_Vars2 or Gfa_Vars!2**
"3" or 2 … n	All local and static variables of the caller (of the caller …) of the caller of the current procedure. Available in Gfa_Tron and Gfa_TronBook proc only.	**Gfa_Vars3, Gfa_Vars4, …, Gfa_Vars9 or Gfa_Vars!n**

When the parameter is the empty string "", a dash "-", or an explicit procedure name, the collection of variables can be created anywhere in the GLL. However, the numbered **Gfa_Vars** collections are relative to the stack and are available only in the **Gfa_Tron** or **Gfa_TronBook** procedures.

Note: To obtain variables at a depth level of more than 9, then you must use the variant that takes a parameter: **Gfa_Vars**("12").

Each instance of a **Gfa_Vars** collection has the following properties:

Property	**Description**
PName	The name of the procedure that is specified as the parameter in the Gfa_Vars() function. This property most interesting for the stack based collections to obtain the name of the procedure that is being executed.
Count	The number of items in the collection.
Item()	A Gfa_Var object for the specified variable.

Availability at design-time

After compiling or performing a syntax-check the **Gfa_Vars** collection is available, as well. The syntax-check (or Run command) collect all variables and the variables are available in the non-stack based **Gfa_Vars**() function. The following piece of code displays the name and type of all global variables in the Debug Output window (don't forget to compile your program first).

Example

```
Sub Gfa_App_V
  Dim v As Gfa_Var, GlobVars As Gfa_Vars
  Set GlobVars = Gfa_Vars("")
  For Each v In GlobVars
    Debug v.Name & #32 & v.Type & #32 & v.TypeName
  Next
EndSub
```

To access the variables of a procedure specify the name (case sensitive) of the procedure as the parameter of **Gfa_Vars**().

```
Set vs = Gfa_Vars!Gfa_App_V      ' using ! notation
```

The name of function may contain a type modifier %, $, &, @, or |. Only ! is forbidden.

```
Set vs = Gfa_Vars!Calculate@     ' variables of
  function Calculate@()
```

See Also

Gfa_Var, Gfa_Types, Gfa_Type

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_Compile and Gfa_DoCompile

Syntax

Gfa_Compile

Gfa_DoCompile

Description

Gfa_Compile compiles and then saves the project as a stand-alone executable (exe), a GFA Editor Extension (.gll), or a GFA-BASIC library (lg32).

Gfa_Compile displays the Compile dialog box to obtain the name of the compiled project, version information and in case of a stand-alone exe its icon.

Gfa_DoCompile compiles the project without displaying the dialog box. However, when a project isn't compiled before (**Gfa_ExeName** = ""), the Compile dialog box is shown, like in **Gfa_Compile**.

Example

```
Sub Gfa_Ex_C
  Gfa_DoCompile
EndSub
```

Remarks

A change in any of the fields of the Compile dialog box changes the project that contains the compiler settings. The **Gfa_Dirty** flag is set. So, after compiling the project must be resaved.

See Also

Gfa_ExeName, Gfa_ExeTime, Gfa_Dirty

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

Trace

Purpose

Debugging command that displays variable values on the output window.

Syntax

[Debug].**Trace** exp

exp:Any evaluation expression.

Description

The **Trace** command is intended for use in debugging and by default works only in the IDE.
Its normal use is to check the value of the variables during debugging.

Example

```
// All output sent to Debug Window
Debug.Show
Local a As Int32, b As Variant
a = 12 : b = 12
Trace a = b         // Prints a = b = True
Trace a             // Prints a = 12
b = Missing
Trace b             // Prints b = (missing)
Trace IsMissing(b)  // Prints IsMissing(b)  = True
```

Remarks

Trace is a shortcut for **Debug**.**Trace**, a method of the **Debug** object like **Assert** and **Print**. By default the **Debug** object is disabled for final executables, but it can be enabled through the Compiler tab in the Properties dialog.

See Also

Assert, Debug

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

Alert Function

Purpose

Draws a message box on the screen.

Syntax

Alert *IconAndFlag, MainText$, DefButton, ButtonText$ [,RetVal]*

RetVal **= Alert**(*IconAndFlag, MainText$, DefButton, ButtonText$*)

IconAndFlag, DefButton	*:*	*iexp*
RetVal	*:*	*ivar*
MainText$, ButtonText$	*:*	*sexp*

Description

An **Alert** box is a special form of a message box. It is used when a point in a program is reached where the program is to be cancelled, a certain branch is to be taken, or some other user decision is to be made.

The first integer expression, *IconAndFlag*, determines which symbol will be included in the **Alert** box together with the message. The following symbols are available:

IconAndFlag	**Meaning**
0	mark symbol
1	stop mark
2	question mark

3	exclamation mark
4	information mark
5	windows flag
6	application mark
7	information mark
16	buttons are placed at the right border
32	shadow
64	text is right aligned
128	text is centered

MainText$ contains the message which is to be displayed in the Alert Box. If the text is too long for one line it can be split in up to 4 lines by using "|".

ButtonText$ contains up to five possible alternatives for user response.

DefButton indicates which of these alternatives the default is. This alternative is then selected by pressing the Return key. The alternatives are numbered from 1 to 5 and are separated from each other by a "|".

RetVal contains the number of the alternative which was actually selected.

Example

```
Auto a$, b$, i%, j%, retval%
OpenW # 1
i% = 2
a$ = "Which procedure should|be executed next"
j% = 1
b$ = "Input | Calculate | Print | File output |
  CANCEL"
retval% = 0
```

```
Alert 2 | 16, a$, j%, b$, retval%
CloseW # 1
```

Creates an Alert Box with a question mark as symbol and the message: "Which procedure should be executed next". The default alternative is "Input". The alternatives are:

Input, Calculate, Print, File output, and CANCEL.

retval% contains the number of the selected alternative.

Remarks

AlertBox is a synonym to **Alert** and can be used instead.

In addition to the menu bar and pop-up menus, the **Alert[Box]** is a third possible way of communication between the program and the user. Furthermore, it can prove useful when incorporated inside LG32 libraries as a customised messagebox, where OCX objects and Dialogs can not be used.

Known Issues

- In Windows 8, 8.1 and 10, the static text box (which holds MainText) and the icon image holder are drawn with white backgrounds; a patch has been created to solve this problem by Sjouke Hamstra and will be released in the near future.
  [Reported by James Gaite, 09/03/2017]

- Alert box does not recognise of multiple monitors and is always displayed on the main monitor. Use Prompt, InputBox or MsgBox instead or, in a GLL, use MsgBox0.
  [Reported by Sjouke Hamstra, 03/04/2018]

See Also

Menu, Popup, Message, MsgBox, Prompt

{Created by Sjouke Hamstra; Last updated: 04/04/2018 by James Gaite}

Accessing HTML Help Files

Introduction

Since the demise of WinHlp32.exe with the advent of Windows Vista, (as of 2015, it was still possible to get a cut-down version of WinHlp32.exe from the Microsoft website for all versions up to and including Windows 8), all Help files have been written using the HTML Help (.chm) format. Unfortunately, many of GFABasic's internal Help calling seems to be hard-wired to direct any WM_HELP message to seek WinHlp32.exe - for example, ShowHelp and the help button added through Message Boxes.

Therefore, this page is dedicated to explain how to call HTML Help files and how to workaround some of the intransigencies of GFABasic in this regard.

APIs, Constants and UDTs Show

Calling the Help File Show

Calling the Help File using the Help Data Type Show

Limitations to Using HTMLHelp Files Show

HTMLHelp Files and Visual Styles Show

Additional Resources Show

{Created by James Gaite; Last updated: 18/03/2018 by James Gaite}

Gfa_hWnd, Gfa_hWndEd, Gfa_Refresh

Window API functions.

Syntax

handle% = **Gfa_hWnd**

handle% = **Gfa_hWndEd**

Gfa_Refresh

Description

Gfa_hWnd returns the window handle of the GFA-BASIC 32 IDE as Long.

Gfa_hWndEd returns the window handle of GFA-BASIC 32 source code editor as Long.

Gfa_Refresh immediate redraw of the client area of the editor. Sometimes it is necessary to redraw the source text before an event sub has finished and the invalidated regions are redrawn.

Example

```
Dim h As Handle = Gfa_hWnd
```

Remarks

Gfa_Refresh is internally implemented as *UpdateWindow*(**Gfa_hWndEd**). A WM_PAINT message is send to the window only if the window's update region (the portion of the window's client area that must be redrawn) is not empty. Normally, when a line is changed, GFA-BASIC 32 adds the area occupied by a line to the **Gfa_hWndEd** window's update region with the *InvalidateRect* function. Eventually, Windows sends the WM_PAINT message when there are no other messages in the application queue for that window.

See Also

Gfa_StatusText

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

Gfa_CopyFile Command and Gfa_InlFileName Function

Syntax

Gfa_CopyFile src$, dest$

$ = **Gfa_InlFileName**(n%)

Description

Copies a (normal) file src$ to a new ':Files' inline resource with its name specified in dest$. The destination name must start with the semi colon : to prevent errors.

```
Gfa_CopyFile "e:\cparse\icodeb.ico", ":icodeb"
```

To delete an entry in the ":Files" tab specify the empty string for src$ and its resource name is dest$.

```
Gfa_CopyFile "", ":icodeb"
```

To copy the data from the inline resource section to a normal file use **CopyFile**.

```
CopyFile ":icodeb", "e:\cparse\icodeb.ico"
```

Note Most GFA-BASIC 32 functions that perform file I/O are capable of loading resources from the inline section directly. These commands and functions parse the filename for a starting colon, and assume an inline resource when found. For instance

```
PlaySound ":dingwav"
```

or

```
Open ":menutext" for Input As # 1
```

Gfa_InlFileName(n%) returns the :File name with specified index n%. When n% (1, 2, ..., n) is larger than the number resources the function returns the empty string.

Remarks

GFA-BASIC 32 supports so called inline files or, otherwise put, it allows for including raw data into the project file. These raw data entities are placed in a packed, mime-encoded data format at the end of the project file.

The inline resources are maintained in the ":Files" tab in the sidebar. When you right click in the client area of the ':Files' tab, a context menu will popup to allow for data insertion by loading a file or by pasting from the clipboard. In addition, when one of the loaded GFA Editor Extensions includes the **Gfa_OnDropInl** event sub, the ':Files' tab will become a drag and drop window and one or more file may be dropped. Inline files can not be used in GLL and Library projects.

The inline files are accessed through the GFA-BASIC 32 **Open For Input As #** statement. Once opened the resources can be read as any other file, GFA-BASIC 32 will decode and unpack on the fly.

When a resource must be used in other functions (API functions for instance), the resource must be copied to disk before it can be used. To copy the file you should use **CopyFile** or **FileCopy**. Usually, the file is copied to a temporary directory and killed after it is used.

Other functions that can load inline resources are **LoadPicture** and **LoadCursor**. Inline resources that are used without GFA-BASIC 32 commands must first be copied out of the inline section to a temporary file on disk before they can be accessed. After using the external file it is then deleted (most often). See LoadBmp.

See Also

Gfa_OnDropInl, CopyFile

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

False Variable

Purpose

0 constant for logical false

Syntax

a? = **False**

Example

```
Local a?, i%
OpenW # 1
i% = 20
If i%
  a? = True
  Print "i% is not equal to 0; a?="; a?
EndIf
i% = 0
If !i%
  a? = False
  Print "i% is equal to 0; a?="; a?
EndIf
```

Prints:

i% is not equal to 0; a?=True

i% is equal to 0; a?=False

See Also

True

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

Array ()= Command

Purpose

Initializes an array from a string.

Syntax

Array ar() = string

ar():any array
string:string expression

Description

This command initializes an integer, floating point, string, or user-defined type array with the contents of a string with elements starting at 0 regardless of whether Option Base is set to 0 or 1; for a string array the input string is split at LF and CRLF characters. (LF = Chr(10) CRLF = Chr(13,10)).

```
Option Base 0
Dim fp$()
Array fp$() = "element 1"#10"element 2"#10"element
  3"#10"element 4"
Print fp$(3) // Prints "element 4"
Option Base 1
Array fp$() = "element 1"#10"element 2"#10"element
  3"#10"element 4"
Print fp$(3) // Also prints "element 4"
```

For a Byte array, **Array ()=** creates a one dimensional array with the same number of elements as the length of the string: each character is copied to an array element and the array has 0..Len(string) - 1 elements.

```
Dim b() As Byte
Array b() = "Testing"
Print CStr(b()) // Prints "Testing"
```

Before any array is created with **Array ()=**, it must first be declared using **Dim** or a similar command; note, however, that it is pointless defining the number of elements through this action as the **Array ()=** automatically re-dimensions the array to accomodate the data you allocate to it.

```
Dim f%(10)
Print Dim?(f()) // Prints 11
Array f() = Mki$(10, 4, 9)
Print Dim?(f()) // Prints 3
```

A user defined type array can also be initialized using this method, although the **Type** must be defined using **Packed** 1.

```
Type TestType Packed 1
  by As Byte
  in As Int
  by2 As Byte
  db As Double
EndType
Dim ar() As TestType
Array ar() = Mk1(1) + Mki(2) + Mk1(3) + Mkd(2.1) +
  Mk1(4) + Mki(5) + Mk1(6) + Mkd(7.1)
Print ar(0).by  // prints 1
Print ar(1).db  // prints 7.1
```

Example

```
OpenW 1
```

```
Global Dim mnu$()
Array mnu$() = "&File"#10 "&New"#10 "&Open"#10
  "&Save"#10 _
  "Save &As"#10 "-"#10 "E&xit"#10 #10 _
  "&Edit"#10 "&Undo"#10 "-"#10 "Copy"#10 "Cut"#10
    "Paste"#10 #10 _
  "&Help"#10 "&About"#10 #10
Menu mnu$()
Do
  Sleep
Until IsNothing(Me)
```

Remarks

Using **Array()=** to initialize an array or user-defined type is particularly useful with editor extensions, because the data can not be stored in **Data** lines.

$ = CStr(a**())**is the reverse of **Array** a**()= $**.

If a() is a byte array, **CStr**() creates a string of length **Dim?**(a|()) with the values of the elements of the array.

If a() is a string array, **CStr**() creates a string by adding all elements of the array and separating them with #13#10 (CRLF).

See Also

CStr()

{Created by Sjouke Hamstra; Last updated: 05/08/2019 by James Gaite}

Object Property

Purpose

Returns an IDispatch reference to a control.

Syntax

object.**Object**

object:OLE Automation, Ocx

Description

The GFA-BASIC 32 Ocx controls provide the **Object** property to obtain an IDispatch interface to the control. The GFA-BASIC 32 Ocx controls support dual interfaces for both early and late binding. When a COM object supports late binding it supports a so called IDispatch interface. When a COM object provides an IDispatch interface, the properties and methods can be executed through a standard function called *Invoke*. Rather than executing a property or method directly, as with early binding, the *Invoke* function takes numerous parameters describing the property or method to call, the possible parameters converted to Variants, an exception info block for returning error information, and some more. Invoke itself must lookup the name of the property or method in the COM library and then call it by its address. Calling *Invoke* for a property or method is a time consuming process, therefore.

Example

```
OpenW 1
```

```
Dim oForm As Object
Set oForm = Win_1.Object
Win_1.AutoRedraw = 1  ' Fast
oForm.AutoRedraw = 1  ' Slow
Do
  Sleep
Until Me Is Nothing

Sub Win_1_OnCtrlHelp(Ctrl As Object, x%, y%)
  ' IDispatch reference to the control.
  Print Ctrl.WhatsThisHelpID // Slow
EndSub
```

Remarks

The **Control** data type is an IDispatch interface as well.

```
Ocx ToolBar tb
Dim ctrl As Control
Set ctrl = tb
```

See Also

Object, _DispId

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

Bchg Function

Purpose

Changes the status of a bit in an integer expression.

Syntax

i = **Bchg**(m, n)(function)

Bchg v, n(command)

m, niexp
v:ivar

Description

Bchg(m, n) sets the n-th bit in the integer expression m to 1 (if this bit is 0), or to 0 (if this bit is 1).

Example

```
OpenW # 1
Dim i% = 10                          //  10 =>
 1010
i% = Bchg(i%, 0) : Print Bin$(i%, 4)  // Prints
 1011
Bchg i%, 1 : Print Bin$(i%, 4)        // Prints
 1001
```

See Also

Bclr(), Bset(), Btst()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Bchg8 Function

Purpose

Changes the status of a bit in a 64 bit integer expression.

Syntax

I = **Bchg8**(m64, n)(function)

~~**Bchg8** v64, n(command)~~

*m, n*iexp
v64:Large var

Description

Bchg8(m, n) sets the n-th bit in the 64-bit integer expression m to 1 (if this bit is 0), or to 0 (if this bit is 1).

Example

```
Print Bin$(10, 4)              // Prints 1010
Print Bin$(Bchg8(10, 0), 4)    // Prints 1011
Dim i64 As Large = 10
i64 = Bchg8(i64, 0)
Print i64                      // Prints 11
```

Remarks

Although listed in the original help file as a command - i.e. Bchg8 v64, n - it seems never to have been implemented as such.

See Also

Bclr8(), Bset8(), Btst8()

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

Sar Function

Purpose

Shifts a bit pattern to the right.

Syntax

% = **Sar**(m, n)32 bit operation

% = **Sar%**(m, n)32 bit operation

Large = **Sar8**(m, n)64 bit operation

&=**Sar&**(m, n)16 bit operation

Word = **Sarw**(m, n) 16 bit operation

| = **Sar|**(m, n)8 bit operation

Description

Sar(m, n) shifts a bit pattern of an integer expression m n steps to the right (Sar = Shift Right), in which the highest bit is copied (and not replaced with zero like with Shr). Each bit shift right is a division by two. An example:

x = -8 : **Sar** x, 3 or -8 **Sar** 3 or **Sar**(-8, 3)

-8 as binary: 1111 1111 1111 1111 1111 1111 1111 0111

Shift: **1**111 1111 1111 1111 1111 1111 1111 1011

Shift: **1**011 1111 1111 1111 1111 1111 1111 1101

Shift: **1**001 1111 1111 1111 1111 1111 1111 1110

Example

```
Debug.Show
Trace Sar(-8, 1) // = -4
Trace Sar(-8, 2) // = -2
Trace Sar(-8, 3) // = -1
Trace -8 Sar 3   // = -1
```

Remarks

Sar can also be used as an operator:

m **Sar** n32-bits operation

m **Sar8** n64 bits operation

See Also

Shr, Shl, Ror, Rol

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

GetByte0, GetByte1, GetByte2, GetByte3 Function

Purpose

The **GetByte** functions read a byte from a 32-bits value.

Syntax

Byte = **GetByte0**(value)

Byte = **GetByte1**(value)

Byte = **GetByte2**(value)

Byte = **GetByte3**(value)

value:aexp of 4-bytes

Description

GetByte0 reads the first byte (MSB - most significant byte) of a value of a 32-bits data type.

GetByte1 reads the second byte of a value (2nd MSB).

GetByte2 reads the third byte of a value (3rd MSB).

GetByte3 reads the fourth byte of a value (LSB - least significant byte).

```
Dim a% = $01020304
Debug.Show
Trace GetByte0(a%)        ' 1
Trace GetByte1(a%)       ' 2
Trace GetByte2(a%)       ' 3
Trace GetByte3(a%)       ' 4
```

Example

Example 1

```
OpenW 1
Local a As Single, byte0%, byte1%, x%
Local byte2%, byte3%
a = 222266 * 456 + 97 * 35786
byte0% = GetByte0(a)
byte1% = GetByte1(a)
byte2% = GetByte2(a)
byte3% = GetByte3(a)
Print a, byte0%, byte1%, byte2%, byte3%
```

Example 2

```
OpenW 1
Local byte_hi0%, byte_hi1%, byte_hi2%, x%
Local byte_hi3%, byte2%, byte3%, a_int%
Local a As Large, byte0%, byte1%
a = _maxLarge
a_int% = HiLarge( a)
byte0% = GetByte0(a)
byte1% = GetByte1(a)
byte2% = GetByte2(a)
byte3% = GetByte3(a)
Print a, byte0%, byte1%, byte2%, byte3%
byte_hi0% = GetByte0(a_int%)
byte_hi1% = GetByte1(a_int%)
byte_hi2% = GetByte2(a_int%)
byte_hi3% = GetByte3(a_int%)
Print a_int%, byte_hi0%, byte_hi1%, byte_hi2%,
  byte_hi3%
```

```
Print Hex$(a)
```

Remarks

GetByte1 is synonymous with **GetBValue**, **GetByte2** with **GetGValue** and **GetByte3** with **GetRValue**.

See Also

GetBValue, GetGValue, GetRValue

{Created by Sjouke Hamstra; Last updated: 07/10/2014 by James Gaite}

SWord Function

Purpose

Sign extension of an unsigned word (Card).

Syntax

% = **SWord**(value)

%	*: signed 32-bit integer*
value	*: card expression*

Description

Expansion of a **Card** to a **Long**. The output value is in the range -32768 to +32767. This will be carried out by the copying of bit 15 (=sign bit) into the bits 16 to 31.

Example

```
OpenW 1
Local a As Card
a = 12345 - 28784
Print Bin$(a, 32)          //
 00000000000000001011111111001001
Print Bin$(SWord(a), 32)   //
 11111111111111111011111111001001
```

Remarks

This function loads the value into the eax register and performs a CDWE assembler instruction to extend the lower 16 bits to the upper 16 bits.

See Also

Byte(), Card(), Short(), UShort(), UWord(), Word()

{Created by Sjouke Hamstra; Last updated: 04/03/2017 by James Gaite}

Exit If Command

Purpose

Serves to terminate a loop when the condition following Exit...If is logically "true".

Syntax

Exit [Do | For] If condition

Condition:any numerical, logical or string condition

Description

The **Exit If** command makes it possible to test and exit any loop for a condition other than the one specified in the loop itself (see For...Next, While...Wend, Repeat...Until and Do...Loop). In contrast to the GoTo command, a loop is terminated in an "orderly" fashion by using **Exit If**.

In other words, **Exit If** always jumps to the first programming statement after the last line of the loop, while GoTo can jump anywhere within a Procedure or Function.

Example

```
OpenW # 1
Dim e% = 1
Dim i% = 1
Do
  e% *= i%
  Print Str$(i%) + "! = "; Str$(e%, 5)
  Exit If e% > 32000
  i% ++
Loop
```

calculates the factorial and stores the result in the variable e%. The calculation is terminated if the result exceeds 32000.

Remarks

The **If** condition **Then Exit Do** (or Loop) command common to other dialects of BASIC can also be used.

See Also

Goto, Exit, Exit Sub

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

On Call Command

Purpose

Branch to one of several specified subroutines, depending on the value of an expression.

Syntax

On n **Call** proc0, proc1, …

n:iexp
proc0, proc1, …:name of a Procedure or Sub to jump to

Description

The value of *n* determines which subroutine is branched to in the procedure list. If the value of *n* is less than 1 or greater than the number of items in the list, then control drops to the statement following **On Call**.

Example

```
OpenW 1
Local a%, n%
n% = 3
On n% Call p1, p2, p3, p4, p5, p6
n% *= 2
On n% Call p1, p2, p3, p4, p5, p6

Procedure p1
  Print "PROCEDURE P1"
Return
```

```
Procedure p2
  Print "PROCEDURE P2"
Return

Procedure p3
  Print "PROCEDURE P3"
Return

Procedure p4
  Print "PROCEDURE P4"
Return

Procedure p5
  Print "PROCEDURE P5"
Return

Procedure p6
  Print "PROCEDURE P6"
Return
```

Remarks

Select Case provides a more structured and flexible way to perform multiple branching.

See Also

SelectCase, On GoTo, On GoSub

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}

P:()(), PasCall() Function

Purpose

executes a subroutine at a specified address and returns a Long value.

Syntax

x = **P:(**addr**)([**parameters]**)**

x = **PasCall(**addr**)([**parameters]**)**

x:iexp
addr:iexp
parameters:aexp

Description

The parameters are placed in reverse order on the stack. **P:()()** and **PasCall()()** expects the subroutine to clear the stack.

a% = P:(addr%)(1, 2, 3)

12[esp] 1

8[esp] 2

4[esp] 3

[esp] Return address

the called routine must end with ret 12, correcting the stack pointer.

The parameters can be coerced to a specific format using by preceding the value with one of the following designators:

Dbl: double
Sng: float, single
Large: Large integer
Cur: Currency value
L: Long
Int: Integer
Var: Variant

Example

```
Dim a% = ProcAddr(test)
~P:(a%)( Large:2, 3 )
' or
~PasCall(a%)( Large:2, 3)

Procedure test(i%, la As Large)
  Print la, i%
EndProc
```

Remarks

A **Procedure** takes it parameters by value using the **StdCall** convention.

See Also

C:(), LC:(), LP:(), Call(), CallX(), CCall(), LCCall(), LPasCall(), StdCall(), LStdCall()

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

EdShowLine Command

Purpose

Shows an arrow in front of a code line.

Syntax

EdShowLine n

Description

EdShowLine can only be used in a subroutine declared with **Tron** *procname*. Although EdShowLine can display the arrow before any line, it is most useful when the current executed line - returned in **TraceLnr** - is marked. Therefore, **EdShowLine** is mostly used together with **TraceLnr**.

Example

```
OpenW 1, 0, 0, 400, 500
Global i%
Tron p
. mov eax, 10
. mov [i%], eax
~1
Troff
CloseW 1

Sub p
  Local j%
  Print "i ="; i; TraceLnr // Trace$
  EdShowLine TraceLnr : Delay .5
  If InStr(Trace$, "[i%]") Then {TraceReg + 7 * 4}
    = 123
EndSub
```

The main program consists of two assembler instructions. The first one moves the value 10 to the register eax, the second moves the contents of eax to the variable i% (the ~1 makes sure, that the last used floating point register is cleared, not relevant here, though.)

The **Tron** procedure p prints the contents of the variable i% followed by the current line number and source code text of that line. The command **EdShowLine** shows the normal **Tron** arrow in front of the actual line. A small delay makes it possible to notice the current line.

Finally, if the source code line contains "[i%]", the value 123 is written as integer into memory, which address is obtained using **TraceReg**+7*4.
As a complete debugger, **Tron** needs access to the processor registers. **TraceReg** returns the address of the memory range, where for the actual processor registers are placed in. With **TraceReg**+7*4 the seventh register (0,1,2,3,4,5,6,eax) will be changed. As a result, 123 will placed in eax and thus in i%.

Remarks

In a compiled program the debugging commands are removed.

See Also

ProcLineCnt(), ProcLnr(), SrcCode$(), Trace, Trace()
TraceLnr, TraceReg, Tron, Troff

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

DmyHmsToDate Function

Purpose

Returns the date for the given day, month, year, hour, minute, and second.

Syntax

dt = **DmyHmsToDate**(d, m, y, h, m, s)

dt: Date expression
d, m, y, h, m, s: iexp

Description

Returns the date for the given day, month, year, hour, minute, and second.

Example

```
OpenW 1
Local x%, dt As Date
dt = DmyHmsToDate(20, 12, 99, 12, 12, 12)
Print dt    // prints: 20.12.99 12:12:12
```

See Also

DmyToDate, HmsToTime

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

DmyToDate Function

Purpose

Returns the date for the given day, month, year.

Syntax

dt = **DmyToDate**(d, m, y)

dt: Date expression
d, m, y: iexp

Description

Example

```
OpenW 1
Local x%, dt As Date
dt = DmyToDate(20, 12, 99)
Print dt      // prints: 20.12.99
```

See Also

DmyHmsToDate, HmsToTime

{Created by Sjouke Hamstra; Last updated: 03/10/2014 by James Gaite}

HmsToTime Function

Purpose

Returns a date for a specified hour, minute, and second.

Syntax

Date = **HmsToTime** (hours, minutes, seconds)

hours, minutes, seconds: iexp

Description

To specify a time, such as 11:59:59, the range of numbers for each **HmsToTime** argument should be in the accepted range for the unit; that is, 0-23 for hours and 0-59 for minutes and seconds. However, you can also specify relative times for each argument using any numeric expression that represents some number of hours, minutes, or seconds before or after a certain time.

Example

```
OpenW 1
Local a As Date, x%
a = HmsToTime(1000000, 120000, 33000)
Print a
// prints: 03/16/78 21:18:16 or 16/03/1878
  21:18:16
```

Remarks

The format of the output can be changed with the using of **Mode Date**..., **Mode Format**..., **Format**....

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 09/10/2014 by James Gaite}

Week Function

Purpose

Returns an Integer specifying a whole number between 1 and 52, inclusive, representing the week of the year.

Syntax

Week(*date***)**

date:Date exp

Description

The function **Week**() returns the week of a Date.

Example

```
OpenW 1
Local z As Date
z = HmsToTime(110000, 20, 4000)
Print z, Week(z)
Print Now, Week(Now)
Print Date, " ", Week(Date)
Print "12/12/1912", " ", Week(#12.12.1912#)
Print FileDateTime("c:\windows\notepad.exe"),
  Week(FileDateTime("c:\windows\notepad.exe"))
```

Remarks

If the date of the last day of the year (or two days in a leap year) is put into this function, the value 1 is returned rather than 53.

See Also

CDate(), Date, Date$, DateAdd(), DateDiff(), DatePart(), DateSerial(), DateTime$(), DateToDmy, DateToDmyHms, DateValue(), Day(), DayNo(), DmyHmsToDate(), DmyToDate(), HmsToTime(), Hour(), IsDate(), Minute(), Month(), Now, Now$(), TimeSerial(), TimeToHms, TimeValue(), Second(), Week(), WeekDay(), Year()

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

CStr(array()) Function

Purpose

Converts a Byte or String array to a string.

Syntax

$ = **CStr(**a()**)**

a():array of any type

Description

If a() is a byte array, **CStr**() creates a string of length **Dim?**(a|()) with the values of the elements of the array.

If a() is a string array, **CStr**() creates a string by adding all elements of the array and separating them with #13#10 (CRLF).

If a() is of any other type, the length of the string is **Dim?**(a()) * **SizeOf**(type).

Example

```
Dim b(0 .. 6) As Byte
b(0) = Asc("T")
b(1) = Asc("e")
b(2) = Asc("s")
b(3) = Asc("t")
b(4) = Asc("i")
b(5) = Asc("n")
b(6) = Asc("g")
```

```
Print CStr(b()) // Prints "Testing"
```

or

```
Dim bw() As Word
Array bw() = "Testing_"
Print Dim?(bw()) // Prints 4
Print CStr(bw()) // Prints "Testing_"
```

Remarks

Array a**()= $** is the reverse of **$ = CStr(**a**())**

See Also

Array ()=

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

_Dc Function

Purpose

Returns the handle of the Device Context for a window area.

Syntax

h=**_Dc**([w%])

h:Handle
w%:integer expression

Description

During its software emulation of multitasking, Windows separates the entire screen into various Device Contexts. These Device Contexts are accessed with handles. If a program is to draw in a particular area of the screen it requires first the handle of the area in question. Various display areas are reordered whenever a new window is opened. For each window a handle can be obtained for the internal display area of the window and another handle for the total window area. **_Dc**(w%) returns the handle of the display area of the internal window area (Client Area). w% is thereby the window number.

Example

```
OpenW # 1
Print _DC(1) // Device Context of Win_1
```

Remarks

Implemented for compatibility reasons.

_Dc(1) is equivalent to **Win_1.hDC**.

_Dc() is equivalent to **Me.hDC**.

See Also

AutoRedraw, hDC, hDC2

{Created by Sjouke Hamstra; Last updated: 20/09/2014 by James Gaite}

Dir Command

Purpose

Prints the contents of a directory.

Syntax

Dir path$ [**To** file$]

path$:sexp;
file$:sexp; optional file name

Description

Dir path$ returns the directories in path name specified in path$. If path$ ends with a ":" or "\", GFA-BASIC automatically appends "*.*". The default destination for the output of the directory is to screen. The specification of **To** file$ is optional. It can be used to redirect the directory output to a file or a peripheral device.

Example

```
OpenW 1
Local x%
Dir "c:\windows\?a.*" To "test.dat"
// if not exist a file will be created and therein
  all file
// names are placed in, which will be found by
  using the
// last pattern.
Dir "c:\windows\*.?st" To "test2.dat"
// or
```

```
Dir "c:\windows\*.lst" To "Test3.dat"
```

See Also

Files[To]

{Created by Sjouke Hamstra; Last updated: 30/09/2014 by James Gaite}

LoadBmp Function

Purpose

Loads a bitmap from file and returns a handle.

LoadBmp() has been superseded by the **LoadImage**() API.

Syntax

h%= **LoadImage**(0, file$, IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE)

h%:bitmap handle
file$:sexp

Description

LoadImage() loads the bitmap specified in file$ into memory. h% is then the handle of the bitmap and can, for example, be used for the **Put** and **Stretch** commands.

The application must call the **FreeBmp** h or the Windows API *DeleteObject*() function to delete each bitmap handle returned by the *LoadImage* function.

Example

```
OpenW 1
Local h As Handle, n As Int32
// Find picture file
Local d$ =
  GetSetting("\\HKEY_CLASSES_ROOT\Applications\GfaW
  in32.exe\shell\open\command", , "")
If Left(d$, 1) = #34 Then d$ = Mid(d$, 2)
n = RInStr(d$, "\") : If n <> 0 Then d$ = Left(d$,
  n - 1)
If Not Exist(d$ & "\gfawintb.bmp") Then _
  MsgBox("Can not locate gfawintb.bmp
    file"#13#10#13#10"Please manually place it in
    the GFABASIC32\Bin folder and try again.")  :
    End
// Create Image OCX and load picture into object
Ocx Image img = "", 10, 10, 356, 16
h = LoadImage(0, d$ & "\gfawintb.bmp",
  IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE)
Set img.Picture = CreatePicture(h, False)
FreeBmp h
Do : Sleep : Until Me Is Nothing
```

Remarks

GFA-BASIC 32 I/O routines are able to read ":Files", because they are implemented to recognize the leading colon as a resource or inline file. API functions don't know about this peculiar GFA-BASIC 32 feature. To use API I/O functions the resource must first copied to a temporary file as in the function below:

```
Function LoadBmp(ByVal FName As String) As Handle
  Try
    If Left$(FName) <> ":"  ' Load from normal file
      LoadBmp = LoadImage(0, FName, IMAGE_BITMAP, 0,
        0, LR_LOADFROMFILE)
    Else     ' copy to Temp directory
      Local path$ = TempFileName("")
      CopyFile FName Over To path$
      LoadBmp = LoadImage(0, path$, IMAGE_BITMAP, 0,
        0, LR_LOADFROMFILE)
      KillTempFile path$
    EndIf
  Catch
    MsgBox "LoadBmp Error #" & Err.Number & #10 &
      Err.Description
  EndCatch
EndFunction
```

File systems attempt to keep all of the data in memory for quicker access rather than flushing the data back to mass storage. A temporary file should be deleted by the application as soon as it is no longer needed.

Another way of loading in a Bitmap from file is to use **LoadPicture**.

See Also

TempFileName, FreeBmp, LoadPicture, Put, Stretch

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

BoundW Function

Purpose

bounds test

Syntax

BoundW[ord](n)

n: integer expression

Description

The **BoundW**(n) function tests if the parameter n fits in a word. This means that when n < -32768 or n >32767 an error message is reported.

Otherwise n is returned unchanged.

Example

```
Try
  Print BoundW(12345)
  Print BoundWord(123456)
Catch
  Print "123456 exceeds capacity"
  MsgBox(Err$)
EndCatch
```

Remarks

The **BoundW**() function serves to find program errors by early discovery of any range violations. **BoundWord**() is a synonym.

See Also

BoundC(), BoundB(), Bound()

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Preset Command

Purpose

Sets a graphic point using the background color.

Syntax

Preset x, y [, color]

Preset [Step] (x, y) [, color]

x, y:Single exp
color:iexp

Description

Preset *x, y, color* sets a graphic point at the coordinates x and y in color *color*. When *color* is omitted the current background color is used. **Preset** can be used as an alternative to:

Color ,**RGB**(r, g, b) : **Plot** x, y

however, Preset will not change the current background color.

Preset x, y or **Preset**(x, y) sets a point in the current foreground color.

Preset Step (x, y) sets a point in the current background color at a distance of x, y from the current position.

Preset Step (x, y), color sets a point in the *color* at a distance of x, y from the current position.

Example

```
OpenW # 1
Do
  DoEvents
  Preset Rand(_X), Rand(_Y), Rand(_C) - 1
Until MouseK %& 2
CloseW # 1
Cls
Color RGB(0, 0, 0), RGB(255, 0, 0)
Do
  DoEvents
  Preset Rand(_X), Rand(_Y)
Until MouseK = 1
CloseW 1
```

Fills the screen slowly with many multicolored points. The second part fills it with red points.

Remarks

Pset sets a point, **Preset** wipes it

See Also

Color, Plot, Draw, Line, SetDraw, Pset, Point, PTst

{Created by Sjouke Hamstra; Last updated: 21/10/2014 by James Gaite}

Field # ...As...At Command

Purpose

Random access file management

Syntax

Field #n, count **As** set$ [, count **As** set$...]

Field #n, count **At**(x) [, count **At**(x), ...]

n:integer expression; channel number
count:integer expression
set$:svar, but not an array variable
x:addr

Description

RANDOM ACCESS files are composed of records and fields. A record is a collection of data, for example an address. A record contains a mixture of fields (the record Address can, for example, be divided into fields: Name, Street, Postcode, and City). Both records and fields have a set size.

Field #n divides records into fields. *n* is the channel number (from 0 to 511) of a file previously opened with **Open**. The integer count defines the corresponding field length. The string variable *set*$ always refers to one field in a record. If a record is divided into several fields, each must be separated with a comma (count **As** *set*$). The sum of individual field sizes must be equal to the length of the record. To save individual fields with length given in *count*, the commands **Lset**, **Rset** and **Mid**$ should be used. Using

Field At numerical variables can be written to an R-file (random access) without having to convert them to strings. The pointer to numerical variables which are to be saved is given in brackets after **At** and the number of bytes to read from this address is given before **At**. A mixture of **As** and **At** is allowed.

The **Field** command can span across several program lines.

The **Field** statement can use a TYPE variable a. The address of the TYPE variable is used.

Field #1, Len(a) At V:a

Example

```
OpenW 1
// a simple declaration
Global i%, name$, town$, zip%, ss$, x%
// to open the file
// Open App.Path & "\addresses.dat" for Random As
  # 1, Len = 64
// Field construct
// Field #1,24 As name$,24 As ss$,4 At (V:zip&)
// Field #1, 12 As town$
//
// or direct with Option Base
Open App.Path & "\addresses.dat" for Random Based
  1 As # 1, Len = 64
Field # 1, 24 As name$
Field # 1, 24 As ss$
Field # 1, 4 At (V:zip%)
Field # 1, 12 As town$
//
For i% = 1 To 5
  Lset name$ = "NAME: " + Str$(i%)
  Lset ss$ = "STREET: " + Str$(i%)
  zip% = i%
  Lset town$ = "TOWN: " + Str$(i%)
  Put # 1, i%
Next
For i% = 1 To 5
  Get # 1, i%
  Print "record_number : "; Str$(i%, 3)
  Print name$
  Print ss$
  Print zip%
  Print town$
Next
Close # 1
Kill App.Path & "\addresses.dat" // Tidy up line
```

See Also

Get#, Put#, Record#

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

ASin Function

Purpose

Returns the arc sine of a numeric expression.

Syntax

# = **ASin**(x#)

Description

ASin(x) expects as function argument x the quotient between the hypotenuse and the side opposite the angle (in a right-angled triangle) and returns the angle in radians. It follows from this that the value of "x" ranges between -1 (equivalent to **Sin**(-PI/2)) and 1 (equivalent to **Sin**(PI/2)).

Example

```
OpenW # 1
Print Asin(-1)        //prints -1.57...
Print Asin(1)         //prints 0.57...
Print Asin(Sin(PI))   //prints 1.22460...
```

Remarks

ASin() is the reverse function of **Sin**() in the range [-pi/2,pi/2].

See Also

Sin(), SinQ(), Cos(), CosQ(), Tan(), Acos(), Atn(), Atan(), Atan2()

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Tanh Function

Purpose

Returns the hyperbolic tangent of a numeric expression.

Syntax

# = **Tanh**(x)

x:aexp

Description

The hyperbolic tangent is defined as the function:

Tanh(x) = (Exp(x)-Exp(-x))/(Exp(x)+Exp(-x)) = (1-Exp(-2*x))/(1+Exp(-2*x))

The function y= **Tanh**(x) returns values between -1 and +1.

Example

```
Debug.Show
Trace Tanh(2.14)         // Prints 0.97269...
Trace Tanh(ArTanH(-0.5))// Prints -0.5
```

Remarks

Tanh() is the reverse function of **ArTanH**().

The hyperbolic cotangent area is obtained with:

CotH(x) = 1 / Tanh(x)

CotH(x) = CosH(x) / SinH(x)

See Also

ArTanH(), SinH(), CosH(), ArSinH(), ArCosH()

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

ArSinH Function

Purpose

Returns the hyperbolic sine area of a numeric expression.

Syntax

# = **ArSinH**(x#)

Description

The hyperbolic sine area is obtained with the function:

ArSinH(x) = Log(x+Sqr(x^2+1))

Example

```
OpenW 1
Local x%
Print ArSinH(2.14)                     // prints
  1.50454...
Print ArSinH(SinH(2.14))               // prints 2.14
Print Log(2.14 + Sqr(2.14 ^ 2 + 1)) // prints
  1.50454...
KeyGet x%
CloseW 1
```

Remarks

ArSinH() is the reverse function of **SinH**().

See Also

SinH(), CosH(), TanH(), ArCosh(), ArTanh()

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

ArCosH Function

Purpose

Returns the hyperbolic cosine area of a numeric expression.

Syntax

# = **ArCosH**(x#)

x:aexp

Description

The hyperbolic cosine area is obtained with the function:

ArCosH(x) = **Log**(x+**Sqr**(x^2-1))

The function y = **ArCosH**(x) returns in y a real number greater than or equal to 0.

Example

```
OpenW # 1
Print ArCosH(2.14)        // Prints 1.39425...
Print ArCosH(CosH(2.14)) // Prints 2.14
```

Remarks

ArCosH() is the reverse function of **CosH**().

See Also

ArSinh(), ArTanh()

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

ArTanH Function

Purpose

Returns the hyperbolic tangent area of a numeric expression.

Syntax

# = **ArTanH**(x#)

Description

The hyperbolic tangent area is obtained with the function:

ArTanH(x) = Log((1+x)/(1-x))/2

Example

```
OpenW 1
Local x%
Print ArTanH(-0.5)                         //
  prints -0.54930...
Print ArTanH(Tanh(-0.5))                   //
  prints -0.5
Print Log((1 + (-0.5)) / (1 - (-0.5))) / 2 //
  prints -0.54930...
KeyGet x%
CloseW 1
```

Remarks

ArTanH() is the reverse function of **TanH**(). The hyperbolic cotangent area is obtained with:

ArCosH(x) = 1/**ArTanH**(x)

See Also

SinH(), CosH(), TanH(), ArSinh(), ArCosh()

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Deg, Rad Functions

Purpose

Used to convert between radians and degrees.

Syntax

d = **Deg**(r)

r = **Rad**(d)

d : aexp; angle in degrees r : aexp; angle in radians

Description

Deg() converts radians to degrees and **Rad**() converts degrees to radians.

Example

```
Debug.Show
Trace Deg(PI / 2)                // Traces 90
Trace Deg(PI)                    // Traces 180
Trace Deg(3 * PI / 2)            // Traces 270
Trace Deg(2 * PI)                // Traces 360
Trace (Rad(90) = PI / 2)         // Traces True
Trace (Rad(180) = PI)            // Traces True
Trace (Rad(270) = 3 * PI / 2)    // Traces True
Trace (Rad(360) = 2 * PI)        // Traces True
```

Remarks

Deg(x) is the reverse function of **Rad**(x), which means:

Deg(**Rad**(PI)) = PI = 3.14...

{Created by Sjouke Hamstra; Last updated: 27/01/2016 by James Gaite}

Random Function

Purpose

Returns a Double random number.

Syntax

# = **Random**(n)

n:Double expression

Description

Returns a 64-bit floating point random number between 0 (inclusive) and n (exclusive). Random is a floating-point operation and takes some more time to execute than **Rand**.

When the numeric expression n is an integer, all numbers have the same probability of being selected, and vice versa. **Random**(n) is equivalent to **Trunc**(**Rnd***n).

Example

```
OpenW # 1
Print Random(10)
```

Prints a random number between 0 and 10.

See Also

Rnd, Rand, Randomize

{Created by Sjouke Hamstra; Last updated: 22/10/2014 by James Gaite}

Fact Function

Purpose

Returns the factorial of a natural number.

Syntax

Fact (n)

n:integer expression

Description

Fact (n) returns the factorial of a natural number n (n!). A factorial is the product of the first n natural numbers, where 0! = 1.

Example

```
OpenW # 1
Print Fact(6) // 6! is 720
```

See Also

Combin(), Variat()

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

. Assembler instruction

Purpose

Invokes the GFA-BASIC 32 Inline Assembler

Syntax

. | **Asm** *mnemonic destination, source*

Description

The dot is a shortcut for the **Asm** keyword and invokes the inline assembler and can appear wherever a GFA-BASIC 32 statement is legal. It cannot appear by itself. It must be followed by an assembly instruction.

The assembler commands use the INTEL parameter sequence, for example:

. mov dest, source

The inline assembler lets you embed assembly-language instructions in your GFA-BASIC 32 programs. The inline assembler is built into the compiler. Inline assembly code can use any variable or function name that is in scope, so it is easy to integrate it with your program’s code.

Example

```
GetRegs
Print _EAX
. mov eax, 1
. inc eax
```

```
GetRegs
Print _EAX
```

The middle two lines can also be written like:

```
Asm mov eax, 1
Asm inc eax
```

Remarks

More about the inline assembler you'll find with Asm.

See Also

Asm

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

% Operator

Purpose

Divides the value of one expression by the value of another, and returns the remainder (modulus).

Syntax

i **%** j

i : avar
j : avar

Description

The modulus, or remainder, operator divides integer number1 by integer number2 and returns only the remainder. The sign of the result is the same as the sign of number1. The value of the result is between 0 and the absolute value of number2.

Example

```
Global l%
Print (42 % 6) //  prints  0
l% = 42 % 5
Print l%       //  prints  2
```

Remarks

% is identical to **Mod**.

See Also

Mod, Fmod, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 17/09/2014 by James Gaite}

Mod8 Operator and Function

Purpose

Calculates the modulo of an integer expression based on a second integer expression.

Syntax

l = i **Mod8** j(operator)
l = **Mod8**(i, j [,m,...])(function)

i, j, m, l:64-bit integer expression

Description

The operator i **Mod8** j and the function **Mod8**(i, j, ...) return a 64-bit integer value. In case one of the parameters isn't an Int64, it is converted to a 64-bit value first (using **CLarge**).

Example

```
Debug.Show
Dim b As Double = 7.1, c As Large, d As Int
Trace b Mod 3             // CLarge(b) Mod 3 = 1
Trace Mod(b, 3)           // CLarge(b) Mod 3 = 1
Trace b Mod8 3.1          // 2 + CLarge(3.1) = 1
Trace b Mod8 4            // 3
Trace Mod8(b, 3)          // b Mod8 3 = 3 - Not
  Correct
Trace Mod8(7, 4, 3)       // 3 - Not Correct
' Mod Command requires an integer variable
c = 42, d = 42
```

```
Mod c, 5 : Trace c        // 5 - Not Correct
Mod d, 5 : Trace d        // 2 - Correct
```

Known Issues

The **Mod8**() function does not appear to work correctly; where possible, use the **Mod8** operator instead.

The type independent **Mod** v, y assignment command doesn't work correctly when v is not an integer.

See Also

Add8, Sub8, Mul8, Div8, +, -, *, /F, \, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

~ Operator

Purpose

a bitwise Not

Syntax

x% = **~** i

i:integer expression

Description

~ i inverts the bit pattern in i.

The one's complement operator, sometimes called the "bitwise complement" or "bitwise NOT" operator, produces the bitwise one's complement of its operand. The operand must be of integral type. This operator performs usual arithmetic conversions; the result has the type of the operand after conversion.

Example

```
Print Bin$(3, 32)     // Prints
  00000000000000000000000000000011
Print Bin$(10, 32)    // Prints
  00000000000000000000000000001010
Print Bin$(~3, 32)    // Prints
  11111111111111111111111111111100
Print Bin$(~10, 32)   // Prints
  11111111111111111111111111110101
```

Remarks

Not is synonymous with ~ and can be used instead.
However, ~ has higher priority so

a% = ~b% + 4 = (Not b%) + 4

a% = ~(b% + 4) = Not b% + 4

See Also

And, Or, Xor, Not, Imp, Eqv, %&, |

{Created by Sjouke Hamstra; Last updated: 20/09/2017 by James Gaite}

Checked, Hidden, Indeterminate, Pressed Properties (Button)

Purpose

Return or set the Button object state.

Syntax

Button.**Checked** [= Boolean]

Button.**Hidden** [= Boolean]

Button.**Indeterminate** [= Boolean]

Button.**Pressed** [= Boolean]

Description

Checked [= ?]	Returns or sets a Boolean that determines the checked state of the button.
Hidden [= ?]	Returns or sets a Boolean that determines the visibility of the button.
Indeterminate [= ?]	Returns or sets a Boolean that determines the indeterminate state of the button (dimmed background).
Pressed [= ?]	Returns or sets a Boolean that determines the pressed state of the button.

Example

```
Ocx ToolBar tb
tb.Add , , "Checked", 1 : tb.Add , , "Hide" :
  tb.Add , , "Indeterminate" : tb.Add , , "Pressed"
tb(1).Checked = True             // Sets the button
  as Checked.  Click to uncheck
tb(3).Indeterminate = True
tb(4).Pressed = True             // Highlights the
  button until clicked.
Do : Sleep : Until Me Is Nothing

Sub tb_ButtonClick(Btn As Button)
  Select Btn.Index
  Case 1
    ' Btn.Caption = (Btn.Checked ? "Checked" :
      "Unchecked")   // Returns 'Not implemented'
      error
  Case 2
    Btn.Hidden = True
  EndSelect
EndSub
```

Remarks

An indeterminate state is a combination of two or more states. For example, if the user selects text in a RichEdit textbox, and some of the text is italicized, the button that represents italicized text cannot be either checked or unchecked; the text in the selection is both. To signify this indeterminate state, set the **Indeterminate** property to True. This dithers the image on the button to create a third state of the button's image.

See Also

Button, ToolBar

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Image, SelectedImage, ExpandedImage Property

Purpose

Returns or sets a value that specifies which **ListImage** object in an **ImageList** control to use with another object.

Syntax

object.**Image** [= *index*]

object.**SelectedImage**[= *index*]

object.**ExpandedImage**[= *index*]

object: Node, Button, Tab

index:An integer or unique string specifying the ListImage object to use with object. The integer is the value of the Index property; the string is the value of the Key property.

Description

Before setting the **Image** property, you must associate an **ImageList** control with a **Toolbar**, **TreeView**, or **TabStrip** control by setting each control's **ImageList** property to an **ImageList** control.

The **SelectedImage** property returns or sets the index or key value of a **ListImage** object in an associated **ImageList** control; the **ListImage** is displayed when a **Node** object is selected. If this property is set to **Null**, the mask of the default image specified by the **Image** property is used.

The **ExpandedImage** property allows you to change the image associated with a **Node** object when the user double-clicks the node or when the **Node** object's **Expanded** property is set to **True**.

Example

```
Local Int m, n
Ocx ImageList iml
iml.ImageHeight = 16 : iml.ImageWidth = 16
iml.Add , "image",
  CreatePicture(LoadIcon(_INSTANCE, 7))
iml.Add , "selimage",
  CreatePicture(LoadIcon(_INSTANCE, 1))
iml.Add , "expimage",
  CreatePicture(LoadIcon(_INSTANCE, 9))
Ocx TreeView tv = "", 10, 10, 200, 300
tv.LineStyle = tvwRootLines
tv.Style = tvwTreelinesPlusMinusPictureText
tv.ImageList = iml
For n = 1 To 10 : Read m
  If m = 0 : tv.Add , , , "Project" & n
  Else : tv.Add m, tvwChild, , "Project" & n
  EndIf
  tv.Node(n).Image = 1
  tv.Node(n).SelectedImage = 2
  tv.Node(n).ExpandedImage = 3
Next n
Data 0,1,1,2,0,0,5,5,6,9
Do : Sleep : Until Me Is Nothing
```

See Also

Node, Button, Tab

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Value Property

Purpose

Returns or sets the value of an object.

Syntax

object.**Value** [= *integer*]

object:Ocx object

Description

Returns the current Value of a Button or Panel object. When **Value** is set a mouse click is executed.

Button - Returns True when the button is pressed. **Value** = True invokes a mouse click.

Panel - Returns True when the **Panel** is clicked. **Value** = True invokes a mouse click.

Example

```
Ocx TextBox tb = "", 10, 10, 45, 15 : .BorderStyle
  = 1 : .ReadOnly = True
Ocx UpDown up : .BuddyControl = tb  : .Max = 1000
  : .Increment = 3 : .Value = 4
Ocx Label lbl = "up.Value = 4", 10, 40, 100, 14
Do : Sleep : Until Me Is Nothing

Sub up_Change
  lbl.Text = "up.Value = " & up.Value
EndSub
```

See Also

Button, Panel

{Created by Sjouke Hamstra; Last updated: 25/10/2014 by James Gaite}

TreeViewName, ListViewName Properties

Purpose

Return the parent Ocx name of a collection item (Tab, Node, and ListItem).

Syntax

Node.**TreeViewName**

ListItem.**ListViewName**

returnvalue: string

Description

TreeViewName returns a string containing the Ocx name of the **TreeView** parent the **Node** belongs to.

ListViewName returns a string containing the Ocx name of the **ListView** parent the **ListItem** belongs to.

Example

```
Local n As Int32
Ocx TreeView tv = "", 10, 10, 200, 300
For n = 1 To 5 : tv.Add , , , "Item" & n : Next n
Ocx ListView lv = "", 220, 10, 200, 300
For n = 1 To 5 : lv.Add , , "Item" & n : Next n
Color 0, RGB(255, 255, 0)
Text 10, 320, "TreeViewName: " &
  tv(1).TreeViewName
```

```
Text 220, 320, "ListViewName: " &
  lv(1).ListViewName
Do : Sleep : Until Me Is Nothing
```

See Also

Node, ListItem, TreeView, ListView

{Created by Sjouke Hamstra; Last updated: 24/10/2014 by James Gaite}

SubItemIndex Property

Purpose

Returns an integer representing the index of the sub item associated with a **ColumnHeader** object in a **ListView** control.

Syntax

*ColumnHeader***.SubItemIndex** [= index%]

Description

Subitems are arrays of strings representing the **ListItem** object's data when displayed in Report view.

The **SubItemIndex** is used to associate the **SubItems** string with the correct **ColumnHeader** object.

The first column header always has a **SubItemIndex** property set to 0 because the small icon and the **ListItem** object's text always appear in the first column and are considered **ListItem** objects rather than subitems.

The number of column headers dictates the number of subitems. There is always exactly one more column header than there are subitems.

Example

```
Global a$, n As Int32
Ocx ListView lv = "", 10, 10, 400, 200 : .View = 3
  : .FullRowSelect = True : .GridLines = True
```

```
For n = 1 To 4 : lv.ColumnHeaders.Add , , "Column"
  & n : lv.ColumnHeaders(n).Alignment = 2 : Trace
  lv.ColumnHeaders(n).SubItemIndex : Next n
For n = 1 To 4
  lv.Add , , ""
  a$ = Rand(10) & ";" & Rand(10) & ";" & Rand(10) &
    ";" & Rand(10)
  lv(n).AllText = a$
Next n
Do : Sleep : Until Me Is Nothing

Sub lv_ColumnClick(ColumnHeader As ColumnHeader)
  // ColumnHeader.SubItemIndex returns 0; use
    ColumnHeader.Index - 1 instead
  Local chi As Int = ColumnHeader.Index - 1, li As
    ListItem
  If lv.SelectedCount <> 0
    Set li = lv.SelectedItem
    Message "Contents of Column" & (chi + 1) & "
      and Row" & li.Index & #13#10 & "is " &
      li.SubItems(chi)
  Else
    Message "Select a row first"
  EndIf
EndSub
```

Known Issues

As shown in the example above, **SubItemIndex** appears to return 0 regardless of the ColumnHeader selected. As a workaround, use *ColumnHeader*.**Index** - 1 instead; this will work as long as the ColumnHeaders appear in the order they were first added to the listview.

See Also

ColumnHeader, ListView

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

SubItems, AllText Property

Purpose

Returns or sets an array of strings (a subitem) representing the **ListItem** object's data in a **ListView** control.

Syntax

ListItem.**SubItems(**index%**)** [= *string*]

ListItem.**AllText** [= *string*]

Description

The *index%* parameter identifies a subitem for the specified **ListItem**.

Subitems are arrays of strings representing the **ListItem** object's data that are displayed in Report view. For example, you could show the file size and the date last modified for a file.

A **ListItem** object can have any number of associated item data strings (subitems) but each **ListItem** object must have the same number of subitems.

There are corresponding column headers defined for each subitem.

You cannot add elements directly to the subitems array. Use the **Add** method of the **ColumnHeaders** collection to add subitems.

AllText returns or sets the text for all subitems of a **ListItem**. The text for each subitem is to be separated by a semicolon.

Example

```
Global a$, n As Int32
Ocx ListView lv = "", 10, 10, 400, 200 : .View = 3
  : .FullRowSelect = True : .GridLines = True
For n = 1 To 4 : lv.ColumnHeaders.Add , , "Column"
  & n : lv.ColumnHeaders(n).Alignment = 2 : Trace
  lv.ColumnHeaders(n).SubItemIndex : Next n
For n = 1 To 4
  lv.Add , , ""
  a$ = Rand(10) & ";" & Rand(10) & ";" & Rand(10) &
    ";" & Rand(10)
  lv(n).AllText = a$
Next n
Do : Sleep : Until Me Is Nothing

Sub lv_ColumnClick(ColumnHeader As ColumnHeader)
  Local chi As Int = ColumnHeader.Index - 1, li As
    ListItem
  If lv.SelectedCount <> 0
    Set li = lv.SelectedItem
    Message "Contents of Column" & (chi + 1) & "
      and Row" & li.Index & #13#10 & "is " &
      li.SubItems(chi)
  Else
    Message "Select a row first"
  EndIf
EndSub
```

See Also

ListItem, ColumnHeaders, ListView

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Bold, Italic, Underline Properties (ListItem, Node)

Purpose

Returns or set the font attribute of the text of the node.

Syntax

object.**Bold** [= Boolean]

object.**Italic** [= Boolean]

object.**Underline** [= Boolean]

object:ListItem, Node object

Description

True sets the attribute and False removes it.

Example

```
Ocx TreeView tv = "", 10, 10, 150, 200
tv.Add , , "Bold" , "Bold"
tv.Add , , "Italic", "Italicised"
tv.Add , , "Underline", "Underlined"
tv("Bold").Bold = True
tv.Node(2).Italic = True
tv.Nodes(3).Underline = True
Do : Sleep : Until Me Is Nothing
```

See Also

ListItem, Node, TreeView, ListView

{Created by Sjouke Hamstra; Last updated: 24/09/2014 by James Gaite}

EnsureVisible Method (ListView, TreeView)

Purpose

Ensures a specified **ListItem** or **Node** object is visible. If necessary, this method expands **Node** objects and scrolls the **TreeView** control. The method only scrolls the **ListView** control.

Syntax

object.**EnsureVisible**

object:ListItem, Node

Description

Use the **EnsureVisible** method when you want a particular **Node** or **ListItem** object, which might be hidden deep in a **TreeView** or **ListView** control, to be visible.

The method returns True if the ListView or TreeView control must scroll and/or expand to expose the object. The method returns False if no scrolling and/or expansion is required.

Example

```
Ocx TreeView tv = "", 10, 10, 100, 300
Dim node As Node
Set node = tv.Add( , , , "David")
Set node = tv.Add(1, tvwChild, , "Mary")
```

```
node.EnsureVisible ' Expand tree to see all nodes.
Do : Sleep : Until Me Is Nothing
```

See Also

Node, ListItem, ListView, TreeView

{Created by Sjouke Hamstra; Last updated: 04/10/2014 by James Gaite}

Ghosted Property

Purpose

Returns or sets a Boolean that determines whether a **ListItem** object in a **ListView** control is unavailable (it appears dimmed).

Syntax

ListItem.**Ghosted** [= Boolean]

Description

The **Ghosted** property is typically used to show when a **ListItem** is cut, or disabled for some reason.

When a ghosted **ListItem** is selected, the label is highlighted but its image is not.

Example

```
Global a$, m As Int, n As Int
Dim li As ListItem
Ocx ListView lv1 = , 10, 10, 500, 150 : lv1.View =
  3
For n = 1 To 5 : lv1.ColumnHeaders.Add , ,
  "Column" & n : Next n
For n = 1 To 5 :
  a$ = "" : For m = 1 To 5 : a$ = a$ & "Item " &
    ((n - 1) * 5) + m & Iif(m <> 5, ";", "") : Next
    m
  lv1.Add , , "" : lv1(n).AllText = a$ : If n = 2
    Then lv1(n).Ghosted = True
```

```
Next n
lv1.FullRowSelect = True
Debug.Show
For Each li In lv1
  Debug li.Text,li.Ghosted
Next
Do : Sleep : Until Me Is Nothing
```

See Also

ListItem, ListView

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

AddFirst, AddLast, AddNext, AddPrev, AddChild Methods (Nodes)

Purpose

Add a **Node** to a **Nodes** collection in a **TreeView** control and returns a reference to the newly created **Node** object.

Syntax

*Nodes***.AddFirst**(relative, key, text, image, selectedimage)

*Nodes***.AddLast**(relative, key, text, image, selectedimage)

*Nodes***.AddNext**(relative, key, text, image, selectedimage)

*Nodes***.AddPrev**(relative, key, text, image, selectedimage)

*Nodes***.AddChild**(relative, key, text, image, selectedimage)

relative, key, text, image, selectedimage: Variant exp

Description

The **Nodes** object supports the **Add** method to add a Node to the collection. The Add method requires the speciation of a relationship with the *relative* node. Rather than using the general **Add** object, GFA-BASIC 32 offers a series of methods that implicitly includes the relationship.

AddFirst	The Node is placed before all other nodes at the same level of the node named in relative.

AddLast	The Node is placed after all other nodes at the same level of the node named in relative. Any Node added subsequently may be placed after one added as Last.
AddNext	The Node is placed after the node named in relative.
AddPrev	The Node is placed before the node named in relative.
AddChild	The Node becomes a child node of the node named in relative.

The arguments of the methods are:

relative	Optional. The index number or key of a pre-existing Node object. The relationship between the new node and this pre-existing node is found in the next argument, relationship.
key	Optional. A unique string that can be used to retrieve the Node with the Item method.
text	Required. The string that appears in the Node.
image	Optional. The index of an image in an associated ImageList control.
selectedimage	Optional. The index of an image in an associated ImageList control that is shown when the Node is selected.

As a **Node** object is added it is assigned an index number, which is stored in the **Node** object's **Index** property. This value of the newest member is the value of the **Node** collection's **Count** property.

Because the **Addxx** methods return a reference to the newly created **Node** object, you can set properties of the new **Node** using this reference.

Example

```
Dim n As Node, nd As Nodes
Ocx TreeView tv = "", 0, 0, 200, 300
.Style = tvwTreelinesPlusMinusText  : .LineStyle =
  tvwRootLines
Set n = tv.AddItem( , , "Bert" , "Bert")
        // Node 1
Set nd = tv.Nodes
// Below are numerous ways to add a Child...
Set n = nd.AddChild("Bert" , "Harry" , "Harry")
       // Node 2
Set n = tv.Nodes.AddChild("Bert", "Charlie",
  "Charlie") // Node 3
Set n = tv.Nodes.Add(2, tvwChild , "Mary" ,
  "Mary")      // Node 4
nd.AddChild "Mary", "Bertha" , "Bertha"
       // Node 5
// Set n = nd.AddFirst("Bert", "Gerald" ,
  "Gerald") doesn't work
/* Use the following instead:
nd.Add "Bert", tvwFirst, , "Gerald" ' or Set n =
  nd.Add("Bert", tvwFirst, , "Gerald")
// Similarly, AddLast, AddNext, AddPrevious do not
  appear to work either
/* Instead use: nd.Add [Relative], [Type], [Key],
  [Text]
tv.Node(5).EnsureVisible
Do : Sleep : Until Me Is Nothing
```

Remarks

As shown in the example above, the **AddFirst**, **AddLast**, and **AddPrev** methods all trigger the 'Invalid Property Value' error report regardless of what values are entered; **AddChild** doesn't seem to be affected by this problem.

GFA-BASIC 32 specific

Instead of explicitly using the **Nodes** collection to access a **Node** element, you can use a shorter notation. First, the **TreeView** supports an **Item** property:

tv.**Item**(idx)tv.**Nodes**.**Item**(idx)

Like the **Item** method of tv.**Nodes, Item** is the default method of **TreeView**. Therefore, a **Node** can be accessed as follows:

tv(idx)tv.**Nodes**(idx)

tv!idxtv.**Nodes**!idx

Each dot saves about 30 bytes of code.

To enumerate over the **Nodes** collection of a **TreeView** Ocx, use For Each on the Ocx control directly, like:

```
Local node1 As Node
For Each node1 In tv : DoSomething(node1) : Next
```

See Also

TreeView, Node, Nodes

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Child, FirstSibling, LastSibling, Previous, Parent, Next, and Root Properties (Node)

Purpose

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object.

Syntax

Node.**Child**

Node.**FirstSibling**

Node.**LastSibling**

Node.**Previous**

Node.**Parent**

Node.**Next**

Node.**Root**

Description

Child - Returns a reference to the first child of a **Node** object in a **TreeView** control.

FirstSibling - Returns a reference to the first sibling of a **Node** object in a **TreeView** control. The first sibling is the **Node** that appears in the first position in one level of a hierarchy of nodes. Which **Node** actually appears in the first position depends on whether or not the **Node** objects at that level are sorted, which is determined by the **Sorted** property.

LastSibling - Returns a reference to the last sibling of a **Node** object in a **TreeView** control. The last sibling is the **Node** that appears in the last position in one level of a hierarchy of nodes. Which **Node** actually appears in the last position depends on whether or not the **Node** objects at that level are sorted, which is determined by the **Sorted** property. To sort the **Node** objects at one level, set the **Sorted** property of the **Parent** node to **True**.

Previous - Returns a reference to the previous sibling of a **Node** object.

Parent - Returns or sets the parent object of a **Node** object. An error occurs if you set this property to an object that creates a loop. For example, you cannot set any **Node** to become a child **Node** of its own descendants.

Next - Returns a reference to the next sibling **Node** of a **TreeView** control's **Node** object.

Root - Returns a reference to the root **Node** object of a selected **Node**.

Example

```
Dim n As Node, nd As Nodes
Ocx TreeView tv = "", 0, 0, 200, 300
.Style = tvwTreelinesPlusMinusText  : .LineStyle =
  tvwRootLines
```

```
Set n = tv.AddItem( , , "Bert" , "Bert")
       // Node 1
Set nd = tv.Nodes
Set n = nd.AddChild("Bert" , "Harry" , "Harry")
      // Node 2
Set n = tv.Nodes.AddChild("Bert", "Charlie",
 "Charlie") // Node 3
Set n = tv.Nodes.Add(3, tvwChild , "Mary" ,
 "Mary")     // Node 4
nd.AddChild "Mary", "Bertha" , "Bertha"
      // Node 5
nd.AddChild "Bert", "Arthur" , "Arthur"
      // Node 6
tv.Node(5).EnsureVisible
'
Debug.Show
Trace tv!Charlie.Child.Text
Trace tv.Nodes("Charlie").FirstSibling.Text
Trace tv.Nodes("Charlie").Next.Text
Trace tv!Charlie.LastSibling.Text
Trace tv!Charlie.Previous.Text
Trace tv(3).Parent.Text
Trace tv!Charlie.Root.Text
Do : Sleep : Until Me Is Nothing
```

Remarks

There are many ways to access a Node element.

NOTE: Caution should be exercised when interrogating the **Child**, **Previous** and **Next** properties: if a referenced node does not exist, a 'Object is Nothing' error is triggered as can be seen if the following line is added to the example above:

```
Trace tv!Harry.Previous.Text
```

See Also

TreeView, Node, Nodes

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Expanded, FullPath Properties (Node)

Purpose

Returns or sets a value that determines whether a **Node** object in a **TreeView** control is currently expanded or collapsed.

Returns the fully qualified path of the referenced **Node** object in a **TreeView** control. When you assign this property to a string variable, the string is set to the **FullPath** of the node with the specified index.

Syntax

Node.**Expanded**[= *boolean*]

Node.**FullPath**

Description

You can use the **Expanded** property to programmatically expand a **Node** object. The following code has the same effect as double-clicking the first **Node**:

When a **Node** object is expanded, the Expand event is generated.

If a **Node** object has no child nodes, the property value is ignored.

The fully qualified path is the concatenation of the text in the referenced **Node** object's **Text** property with the **Text** property values of all its ancestors. The value of the **PathSeparator** property determines the delimiter.

Example

```
Ocx TreeView tv = "", 10, 10, 200, 400
tv.LineStyle = tvwRootLines
tv.Style = tvwPlusMinusText
tv.Add , , , "David"
tv.Add 1, tvwChild, , "Mary"
tv.Add 1, tvwChild, , "Harold"
tv.Add 1, tvwNext, , "Mildred"
tv.Add 4, tvwChild, , "Jennifer"
tv.Nodes(1).Expanded = True
tv(4).Expanded = True
Print AT(40, 1); "Harold's path: "; tv(3).FullPath
Do : Sleep  : Until Me Is Nothing
```

Remarks

A Node can be accessed in several different ways.

See Also

Node, TreeView, Expand

{Created by Sjouke Hamstra; Last updated: 05/10/2014 by James Gaite}

Draw Method

Purpose

Draws an image into a destination device context, after performing a graphical operation on the image.

Syntax

ListImage**.Draw**(hDC [,x] [,y] [,style])

x, y, style:Variant

Description

Draws an image into a destination device context *hDC* , at *x, y*, and with *style*.

Style	Meaning
0	(Default) Normal. Draws the image with no change.
1	Transparent. Draws the image using the **MaskColor** property to determine which color of the image will be transparent.
2	Selected. Draws the image dithered with the system highlight color.
3	Focus. Draws the image dithered and striped with the highlight color creating a hatched effect to indicate the image has the focus.

Example

```
OpenW 1, 30, 30, 300, 300 : AutoRedraw = 1
Cls 2
Ocx ImageList iml
iml.ImageWidth = 32
iml.ImageHeight = 32
iml.ColorFormat = 0
iml.MaskColor = colBtnFace
iml.UseMaskColor = True
iml.BackColor = colBtnFace
iml.ListImages.Add , "GFA",
  CreatePicture(LoadIcon(_INSTANCE, 1), False)
iml.ListImage(1).Draw Win_1.hDC, 40, 40, 1
Do : Sleep : Until Win_1 Is Nothing
```

See Also

ImageList, ListImages

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

ExtractIcon Method

Purpose

Creates an icon from the bitmap in the **ListImage** object.

Syntax

ListImage.**ExtractIcon**()

Description

Creates an icon from the bitmap in the **ListImage** object and returns a reference to the newly created icon as a Picture object.

Example

```
Dim p As Picture
Ocx ImageList iml
iml.ImageHeight = 32 : iml.ImageWidth = 32 :
  iml.MaskColor = $0c0c0c
iml.Add , "warning", CreatePicture(LoadIcon(Null,
  IDI_WARNING), False)
OpenW 1
Ocx Label lbl = "This is a no-go area", 10, 10,
  100, 100 : lbl.Alignment = 2
Ocx Image img = "", 15, 15, 32, 32 :
  img.Transparent = True
Set img.Picture = iml.ListImages(1).ExtractIcon
Ocx Command cmd = "Do not click", 10, 120, 100, 22
cmd.MousePointer = basCustom : Set cmd.MouseIcon =
  iml.ListImages(1).ExtractIcon
```

```
lbl.MousePointer = 99 : Set lbl.MouseIcon =
  iml.ListImages(1).ExtractIcon
Do : Sleep : Until Win_1 Is Nothing
```

See Also

ListImage, ListImages, ImageList

{Created by Sjouke Hamstra; Last updated: 10/10/2014 by James Gaite}

Mode Function

Purpose

Returns the different options for string conversions and comparisons.

Syntax

$ = **Mode**(option)

Description

Returns the current **Mode** settings.

Mode(**BaseYear**) - A 4 character string, default "1930".

Mode(**Format**) - A 4 character string with the Format settings.

Mode(**Val**) - A 1 character string with the decimal separation character.

Mode(**Using**) - A 2 character string with the **Using** settings.

Mode(**Date**) - A 1 character string with the Date$() separation character.

Mode(**Compare**) - A string with the Compare setting ("Text","Binary", or a number).

Mode(**All**) - A string with all **Mode** option settings. Internal format is undocumented.

Mode(**StrSpace**) - A 1 character string with the **StrSpace** settings ("0" or "1").

Mode(**Lang**) - A 3 character string with the language setting.

Mode also returns some operating settings.

Mode(**Language**) - OS language (in the Netherlands: "Nederlands (Nederland)").

Mode(**Language Eng**) - OS language in English (in the Netherlands: "Dutch").

Mode(**Language Native**) - OS language in native (in the Netherlands: "Nederland").

Mode(**Country**) - OS country (in the Netherlands: "Nederland").

Mode(**Country Eng**) - OS country in English (in the Netherlands: "Netherlands").

Mode(**Country Native**) - OS country (in the Netherlands: "Nederland").

Mode(**Ctry**) - OS country in short (in the Netherlands: "NLD").

Mode(**Ctry Code**) - OS country code (in the Netherlands: "31").

Mode(**Lang List**) - Lists the short name for the available countries for the OS.

Example

Display all Mode settings:

```
Debug.Show
Trace Mode(Format)
Trace Mode(Val)
Trace Mode(Using)
Trace Mode(Date)
Trace Mode(Compare)
Trace Mode(All)
Trace Mode(StrSpace)
Trace Mode(Lang)
Trace Mode( Language)
Trace Mode( Language Eng)
Trace Mode( Language Native)
Trace Mode( Country)
Trace Mode( Country Eng)
Trace Mode( Country Native)
Trace Mode( Ctry)
Trace Mode( Ctry Code)
```

A list of the available countries:

```
Local a As String, i As Int, x%
a = Mode(Lang List)
For i = 1 To Len(a) Step 4
  Mode Lang (Mid$(a, i))
  Debug.Print Left$(Mode(Lang), 3); #9, Mode(
    Language)
Next
Debug.Show
```

See Also

Mode, Using, Format, Str, Date$, Time$, Val

{Created by Sjouke Hamstra; Last updated: 18/10/2014 by James Gaite}

Clear Command

Purpose

Deletes all variables.

Syntax

Clear v1[,v2,...]

v1, v2, ... : variables

Description

This command cannot be used inside loops or subroutines. A **Clear** is performed automatically when the program starts up. For arrays use **Erase**.

Example

```
Local Int32 x = 2, y = 2
OpenW # 1
Print x, y       // Prints 2 2
Clear x, y
Print x, y       // Prints 0 0
```

Remarks

Synonymous with **Clear** you can use **Clr**.

See Also

Clr, Erase

{Created by Sjouke Hamstra; Last updated: 25/09/2014 by James Gaite}

Key Codes and ASCII Values

The following list includes values for the most common Key Codes (also known as Scan or Virtual Key Codes) used with **KeyDown**, **KeyUp** and **Screen_KeyPreview** events and ASCII/ANSI codes for the first 256 characters (used with **KeyPress**).

For a full list of Virtual Key Codes see MSDN.

Control Characters

Show

Other Non-Character Keys

Show

NumPad Codes

Show

Characters in the ASCII table

Show

Windows 1252 ANSI Codes

Show

Conversion Code

The following simple but clever bit of code converts virtual key codes to ASCII and comes from this page on Sjouke

Hamstra's blog.

```
' Press shift-key than click mouse
Debug.Show
Trace Chr(VkKeyToAscii(65))
Trace Chr(VkKeyToAscii(Asc("8"))) ' -> *

Function VkKeyToAscii(keycode As Int) As Int
  // Sjouke Hamstra
  Dim sb As String * 4
  Static Dim keyboardState(256) As Byte
  ~GetKeyboardState(ArrayAddr(keyboardState()))
  If ToAscii(keycode, 0,
    ArrayAddr(keyboardState()), sb, 0) == 1
    Return Asc(sb)
  Else
    Return 0
  EndIf
EndFunc
```

{Created by Sjouke Hamstra; Last updated: 15/12/2015 by James Gaite}

Multithreading with GB32

Introduction Show

What is a Thread? Show

When to use Multithreading Show

How Many Threads? Show

Creating and Terminating Threads Show

Passing Values using Parameters Show

Suspending and Resuming Threads Show

Passing Messages Between Threads Show

Assigning a Thread to a Specific Core Show

Creating a Timer Thread Show

Restricting and Controlling Thread Access Show

Help with Debugging Show

{Created by James Gaite; Last updated: 08/03/2018 by James Gaite}

WinVer Function

Requires: gfawinx.lg32

Purpose

Retrieves version information about the currently running operating system.

Syntax

ret = WinVer([IsWindows])

ret : integer or boolean value

Description

When **WinVer** does not specify a value in its argument, WinVer returns a Long with the OS version in hexadecimal format. The return value can be displayed using the Hex() function. The following values are possible.

```
0x0400 // Windows NT 4.0
0x0500 // Windows 2000
0x0501 // Windows XP
0x0502 // Windows Server 2003
0x0600 // Windows Vista
0x0600 // Windows Server 2008
0x0601 // Windows 7
0x0602 // Windows 8
0x0603 // Windows 8.1
0x0A00 // Windows 10
```

However, Microsoft wants us to abandon the old way of obtaining the OS version and wants us to use the newer version-helpers from the VersionHelpers.h SDK file. **WinVer** implements the version helper functions and wraps them into a single function. To identify the current OS use one of the following self-explanatory constants for the parameter of **WinVer**:

```
IsWindowsXPOrGreater
IsWindowsXPSP1OrGreater
IsWindowsXPSP2OrGreater
IsWindowsXPSP3OrGreater
IsWindowsVistaOrGreater
IsWindowsVistaSP1OrGreater
IsWindowsVistaSP2OrGreater
IsWindows7OrGreater
IsWindows7SP1OrGreater
IsWindows8OrGreater
IsWindows8Point1OrGreater
IsWindowsThresholdOrGreater
IsWindows10OrGreater
IsWindowsServer
```

With the above constants, **WinVer** returns 0 (False) if the application isn't running on the requested OS (or greate if applicabler) and −1 (True) if it is.

Example

```
$Library "gfawinx"
Debug.Show
Debug Hex(WinVer())
If WinVer(IsWindows10OrGreater) Then Debug.Print
  "Running on Windows 10"
```

Remarks

WinVer returns false (0) when called by applications that do not have a compatibility manifest for Windows 8.1 or Windows 10 even if the application is running on one of these OSes. The GFA-BASIC 32 version 2.57 includes a compatibility manifest so that functionality of Windows 8.1 and Windows 10 are ‘unlocked’ and WinVer will return True.

For more information see here

See Also

WinVersion.

{Created by Sjouke Hamstra; Last updated: 13/08/2019 by James Gaite}

Window Messages - Keyboard Input

The following Window Messages (WM_) are raised as a result of input through the keyboard (or, sometimes, the mouse).

WM_CHAR | WM_DEADCHAR | WM_HOTKEY | WM_KEYDOWN | WM_KEYUP | WM_SYSCHAR | WM_SYSDEADCHAR | WM_SYSKEYDOWN | WM_SYSKEYUP

WM_CHAR $0102 (258)

Posted to the window with the keyboard focus when a WM_KEYDOWN message is translated by the TranslateMessage function.

wparam value:
The character ASCII Code of the key pressed.

lparam value:

Bits 0-15	The Repeat Count - the number of times the keystroke is repeated due to the user keeping the key depressed.
Bits 16-23	The Scan Code‡ of the key pressed.
Bit 24	Set if Extended Key* is pressed.
Bits 25-28	Reserved.
Bit 29	Set if ALT key pressed at the same time.
Bit 30	Set if this is a repeat of a previous key press.

Bit 31	Set if the key is being released.

WM_DEADCHAR $0103 (259)

A character code generated by a dead key which is posted to the window with the keyboard focus when a WM_KEYUP message is translated by the TranslateMessage function. A dead key is a key that generates a character, such as the umlaut (double-dot), that is combined with another character to form a composite character. For example, the umlaut-O character (Ö) is generated by typing the dead key for the umlaut character, and then typing the O key.

wparam value:
The character ASCII Code of the key pressed.

lparam value:

Bits 0-15	The Repeat Count - the number of times the keystroke is repeated due to the user keeping the key depressed.
Bits 16-23	The Scan Code‡ of the key pressed.
Bit 24	Set if Extended Key* is pressed.
Bits 25-28	Reserved.
Bit 29	Set if ALT key pressed at the same time.
Bit 30	Set if this is a repeat of a previous key press.
Bit 31	Set if the key is being released.

WM_HOTKEY $0312 (786)

Posted when the user presses a hot key registered by the RegisterHotKey function.

wparam value:
The identifier of the hot key that generated the message. If the message was generated by a system-defined hot key, this parameter will be one of the following values: IDHOT_SNAPDESKTOP (-2) or IDHOT_SNAPWINDOW (-1).

lparam value:

Bits 0-15	The Scan Code‡ of the non-'hotkey' pressed.
Bits 16-31	The hotkey(s) defined as: **MOD_ALT**($0001) - Either of the Alt keys was pressed. **MOD_CONTROL**($0002) - Either of the Ctrl keys was pressed. **MOD_SHIFT**($0004) - Either of the Shift keys was pressed. **MOD_WIN**($0008) - Either of the Windows keys was pressed.

WM_KEYDOWN $0100 (256)

Posted to a window when a non-system key† is pressed. **WM_CHAR** can be used instead to return the character ASCII/ANSI code.

wparam value:
The character Key Code of the key pressed.

lparam value:

Bits 0-15	The Repeat Count - the number of times the keystroke is repeated due to the user keeping the key depressed.
Bits 16-23	The Scan Code‡ of the key pressed.
Bit 24	Set if Extended Key* is pressed.
Bits 25-28	Reserved.
Bit 29	Always reset or 0.
Bit 30	Set if this is a repeat of a previous key press.
Bit 31	Always reset or 0.

WM_KEYUP $0101 (257)

Posted to a window when a non-system key† is released.

wparam value:
The character Key Code of the key pressed.

lparam value:

Bits 0-15	The Repeat Count - the number of times the keystroke is repeated due to the user keeping the key depressed.
Bits 16-23	The Scan Code‡ of the key pressed.
Bit 24	Set if Extended Key* is pressed.
Bits 25-28	Reserved.
Bit 29	Always reset or 0.
Bit 30	Always set or 1.
Bit 31	Always set or 1.

WM_SYSCHAR $0106 (262)

The product of WM_SYSKEYDOWN being passed through the TranslateMessage function, this returns details of the system key† which was pressed.

wparam value:
The character ASCII Code of the key pressed.

lparam value:

Bits 0-15	The Repeat Count - the number of times the keystroke is repeated due to the user keeping the key depressed.
Bits 16-23	The Scan Code‡ of the key pressed.
Bit 24	Set if Extended Key* is pressed.
Bits 25-28	Reserved.
Bit 29	Set if ALT key pressed at the same time.
Bit 30	Set if this is a repeat of a previous key press.
Bit 31	Set if the key is being released.

WM_SYSDEADCHAR $0107 (263)

Sent to the window with the keyboard focus when a WM_SYSKEYDOWN message is translated by the TranslateMessage function. WM_SYSDEADCHAR specifies the character code of a system dead key — that is, a dead key that is pressed while holding down the ALT key.

wparam value:
The character ASCII Code of the key pressed.

lparam value:

Bits 0-15	The Repeat Count - the number of times the keystroke is repeated due to the user keeping the key depressed.
Bits 16-23	The Scan Code‡ of the key pressed.
Bit 24	Set if Extended Key* is pressed.
Bits 25-28	Reserved.
Bit 29	Set if ALT key pressed at the same time.
Bit 30	Set if this is a repeat of a previous key press.
Bit 31	Set if the key is being released.

WM_SYSKEYDOWN $0104 (260)

Posted to a window when a system key† is pressed. **WM_SYSCHAR** can be used instead to return the character ASCII/ANSI code.

wparam value:
The character Key Code of the key pressed.

lparam value:

Bits 0-15	The Repeat Count - the number of times the keystroke is repeated due to the user keeping the key depressed.
Bits 16-23	The Scan Code‡ of the key pressed.
Bit 24	Set if Extended Key* is pressed.
Bits 25-28	Reserved.
Bit 29	Set if ALT key pressed at the same time.
Bit 30	Set if this is a repeat of a previous key press.

Bit 31	Always reset or 0.

WM_SYSKEYUP $0105 (261)

Posted to a window when a system key† is released.

wparam value:
The character Key Code of the key pressed.

lparam value:

Bits 0-15	The Repeat Count - the number of times the keystroke is repeated due to the user keeping the key depressed.
Bits 16-23	The Scan Code‡ of the key pressed.
Bit 24	Set if Extended Key* is pressed.
Bits 25-28	Reserved.
Bit 29	Set if ALT key pressed at the same time.
Bit 30	Always set or 1.
Bit 31	Always set or 1.

* Extended Keys are: the right ALT and the right CTRL keys on the main section of the keyboard; the INS, DEL, HOME, END, PAGE UP, PAGE DOWN and arrow keys in the clusters to the left of the numeric keypad; and the divide (/) and ENTER keys in the numeric keypad.

† A system key event is triggered either by pressing F10, having the Alt key held down while pressing another key or when no window currently has the keyboard focus.

‡ The code for the actual key, not character, pressed on the keyboard which is then translated into a key code or ASCII

value.

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Activate, Deactivate Methods

Purpose

Activate - activates a form, bringing it to the foreground

Deactivate - deactivates a form, bringing it to the background.

Syntax

*Form.***Activate( )**

*Form.***Deactivate( )**

Description

Activate causes the currently selected component to be activated as if it were clicked.

Example

```
Form test
Ocx Command cd = "Deactivate", 10, 10, 100, 22
Do
  Sleep
Until Me Is Nothing

Sub cd_Click
  test.Deactivate
EndSub
```

See Also

Form, GotFocus, LostFocus, SetFocus, Activate, Deactivate

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

LPrint Command and LPos Function

Purpose

Prints a string on the current active printer and returns the current virtual column number within the printer object.

Syntax

Lprint p$

x = **LPos**(y)

x:integer y:dummy value

Description

Prints the string plus a CRLF on the printer of the **Printer** object. This command allows the printer to be used as a line printer.

A new page is automatically generated at the end of the printable area.

Example

```
SetPrinterByName "Microsoft XPS Document Writer"
Output = Printer
FontSize = 12
FontName = "courier new"
Printer.StartDoc "Test"
Printer.StartPage
Lprint "Hello"
```

```
Trace LPos(0)   // Returns 1 due to the implied
  CRLF added by LPrint
Lprint "Hello ";
Trace LPos(0)   // Returns 7 due to the semi-colon
  cancelling the CRLF
Lprint "John"
Trace LPos(0)   // Returns 1 again due to the
  implied CRLF added by LPrint
Printer.EndPage
Printer.EndDoc
Debug.Show
```

Remarks

Lprint is synonymous to

```
Output = Printer
Print
```

You can use **Lprint** without the using of **StartDoc** and **StartPage**; if needed **Lprint** creates itself, as well as **EndPage** and **EndDoc**.

```
SetPrinterByName "Microsoft XPS Document Writer"
Lprint "This is a test"
Printer.ForeColor = RGB(255, 0, 0)
Lprint "This is a test"
Printer.ForeColor = RGB(0, 255, 0)
Lprint "This is a test"
Printer.ForeColor = RGB(0, 0, 255)
Lprint "This is a test"
Printer.ForeColor = RGB(255, 255, 0)
Lprint "This is a test"
Printer.ForeColor = RGB(255, 0, 255)
Lprint "This is a test"
Printer.ForeColor = RGB(0, 255, 255)
Lprint "This is a test"
```

The printer is not initialized before **StartDoc**. Since Lprint implicitly executes a StartDoc, the printer can be initialized by **Lprint "";**

However, as can be seen from the above example, by initialising the printer with **LPrint**, the fontsize to pixel ratio remains as per the screen rather than the printer and the resulting print is miniscule. This can only be remedied by inserting **Printer.Fontsize** statement which can not be done before the output has been switched to the printer (otherwise an 'Unspecified Error' is raised). To get around this problem, the following lines should be placed before the first **LPrint** statement:

```
Lprint ""; // The semi-colon keeps the vitual
  cursor at the top-left of the screen
// or just use Output = Printer
Printer.FontSize = 10
```

See Also

Printer

{Created by Sjouke Hamstra; Last updated: 12/10/2014 by James Gaite}

Message Event

Purpose

Occurs when a message for a **Form** is retrieved from the message queue.

Syntax

Sub *Form*_**Message**(hWnd%, Mess%, wParam%, lParam%)

Description

Sleep and **DoEvents**, but **GetEvent** and **PeekEvent** also, retrieve messages from the application message queue. Before the retrieved message is handled and dispatched to the window procedure of a window, the **Message** event sub is invoked. The **Message** event sub is only executed for the messages that are posted to the message queue, these include WM_PAINT, WM_MOVE, WM_SIZE, WM_COMMAND, WM_SYSCOMMAND, WM_CHAR, and WM_KEY*. Most of these messages have an accompanying sub event (Form_Paint, Form_Moved, Form_ReSize, Form_MenuEvent, Form_SysMenuOver, Form_Key*, etc. Before these event subs are executed GFA-BASIC 32 invokes the **Message** event sub passing the message number and its parameters.

One of the messages that is of interest is the WM_COMMAND message from standard (non-ocx) controls. This is message specifies exactly what happened with a control. The WM_COMMAND message is a good candidate to process in a **Message** event sub. It could also be handled in the **MessageProc** event sub, but this is a *real* callback subroutine allowing to handle messages at the lowest level.

A big disadvantage of a call back procedure is the lack of debug capabilities. A Try/Catch handler is necessary to catch errors. The **Message** event sub is more robust.

Example

```
Dlg 3D On
Global style%, style2%, File$
Dlg Base Unit
style% = WS_BORDER | WS_TABSTOP
style2% = BS_DEFPUSHBUTTON | WS_TABSTOP
Dialog # 1, 10, 10, 150, 100, "Test-Dialog"
  RichEditCtrl "", 101, 50, 10, 80, 14, style%
  PushButton "OK", IDOK, 10, 60, 40, 14, style2%
  PushButton "CANCEL", IDCANCEL, 80, 60, 40, 14,
    style2%
EndDialog
ShowDialog # 1
// to fill the edit field
File$ = "GFA-User"
_Win$(Dlg(1, 101)) = File$
Do
  Sleep
Until Me Is Nothing

Sub Dlg_1_Message(hWnd%, Mess%, wParam%, lParam%)
  Select Mess
  Case WM_COMMAND
    Select wParam
    Case IDOK
      File$ = _Win$(Dlg(1, 101))
      CloseDialog # 1
      OpenW 1
      Print File$ : Print
      Print "End with Alt + F4"
    EndSelect
  EndSelect
```

```
EndSub
```

Remarks

You can easily test in which order the sub events are called. For posted messages, those that are retrieved from the message queue using Sleep, the **Message** event is called before any other sub. Then the message is dispatched to the window procedure and the **MessageProc** is called. And finally, the event sub is invoked. For a WM_SIZE message the sequence is:

Win_1_Message()
Win_1_MessageProc()
Win_1_ReSize

See Also

MessageProc, DDEWndProc, Form

{Created by Sjouke Hamstra; Last updated: 17/10/2014 by James Gaite}

Gfa_LineCnt and Gfa_TopLine Function

Syntax

n% = **Gfa_LineCnt**

line% = **Gfa_TopLine**

Description

Gfa_LineCnt Returns the number of lines of the program.

line% = **Gfa_TopLine** returns the line currently at the top of the editor window.

Although the code for the **Gfa_TopLine=** assignment is present, a bug prevents its use.

Example

An easy workaround is

```
' emulate Gfa_TopLine=

Gfa_Line = 1

Gfa_Line line%
```

See Also

Gfa_Line

{Created by Sjouke Hamstra; Last updated: 12/05/14 by James Gaite}

Gfa_ExeName and Gfa_ExeTime

Project file information.

Syntax

$ = **Gfa_FileName**

date = **Gfa_FileTime**

$ = **Gfa_ExeName**

date = **Gfa_ExeTime**

Description

Gfa_FileName returns the full path and filename of the current project. If the project has no name, when it isn't saved before, this function returns an empty string.

Gfa_FileTime returns the file date of the latest save action of the project currently loaded. The return value is of type Date. In case of an error the return value is CDate(0.0).

Gfa_ExeName returns the name of the compiled project currently loaded in the IDE. This function can be used to determine whether a program is compiled before, if it isn't the function returns an empty string.

Gfa_ExeTime returns the file date of the compile Exe, GLL, or lg32. The return value is of type Date. In case of an error the return value is CDate(0.0).

Example

Insert the filename and time, exe name, and time.

```
Sub Gfa_App_F
  Dim i% = PopUp(" FileName| FileTime| Exe
    FileName| Exe FileTime")
  Gfa_Insert Choose(i% + 1, Gfa_FileName,
    Gfa_FileTime, Gfa_ExeName, Gfa_ExeTime)
EndSub
```

See Also

Gfa_Compile, Gfa_DoCompile

{Created by Sjouke Hamstra; Last updated: 08/10/2014 by James Gaite}

OnHelp Event (CommDlg)

Purpose

Occurs when the Help button on a common dialog is selected.

Syntax

Sub *CommDlg*_**OnHelp**

Description

If you've created a Help file for your application you can use **ShowHelp** to invoke WinHelp, or use the example in Acessing HTMLHelp Files for .chm files, to display help.

Example

```
OpenW 1
Ocx CommDlg cd
cd.Flags = cdfScreenFonts | cdfShowHelp
cd.ShowFont
CloseW 1

Sub cd_OnHelp
  Me.Caption = "Help Requested"
EndSub
```

See Also

CommDlg, ShowHelp

{Created by Sjouke Hamstra; Last updated: 16/07/2015 by James Gaite}

The Manifest File and Common Controls

Common Controls

GFABASIC32 comes with many OCX Controls to allow input and output and these controls are underpinned by Windows' Common Controls library.

From Windows XP onwards, Microsoft allowed access to two versions of its Common Controls library: version 5 which retained the appearance and functionality found in Windows 98/ME; and version 6, originally termed as XP Styles, which matches the controls to the themes and styles of whichever version of Windows a program is run on, as well as adding additional functionality.

By default, Windows assumes any program will run using version 5, and GFABASIC32 is no different. If either the IDE or any program compiled by GFABASIC32 is run, all OCX controls will appear as they did in Windows 98/ME.

Manifest Files

To change the styles to those of Common Controls version 6, it is necessary to use a **Manifest File**. Manifest files can perform numerous tasks, but the file which is supplied with GFABASIC32 (see GfaWin32.exe.manifest) has only one task: to tell Windows to use Coomon Controls version 6 rather than the default version 5.

To effect this, the Manifest file can either be in stand-alone form or embedded in the executable program itself.

- To create a stand-alone file, simply make a copy of the GFABASIC32's Manifest file, paste it into the folder which contains the executable file you wish to affect and change the Manifest's name to suit. For example, if you compile a program called 'program.exe', you would copy the GfaWin32.exe.manifest file into the same file as the compiled file and rename it 'program.exe.manifest'.
- From IDE version 2.40 onwards, there is an option on the Compile form (Project -> Compile/Build) to 'Add Manifest Resource'; this embeds a Manifest file into the executable, doing away with the need to create a stand-alone file. For those with earlier versions of the IDE, Peter Heinzig created a program here which does exactly the same thing.

Known Issues

There are a number of known issues where using a Manifest file does not make the desired transition from version 5 to version 6 of the Common Controls library. Some of the more common ones are:

1. [Stand-alone only] If there a spaces in the filename of the executable - and thus in the accompanying stand-alone Manifest file - this can cause a failure; replacing the spaces in both files with underscores ('_') usually fixes this problem.
2. Occasionally after a change to a compiled file or a Windows Update, controls can return to version 5; simply restarting the computer can, occasionally fix this problem.

There are other issues which can arise with using Manifest files - they are generally Windows-wide rather than just GFABASIC32-specific - and they are dealt with in more detail in Sjouke Hamstra's blog here and here.

{Created by James Gaite; Last updated: 03/03/2018 by James Gaite}

SinH Function

Purpose

Returns the hyperbolic sine of a numeric expression.

Syntax

# = **SinH**(x)

x:aexp

Description

The hyperbolic sine is defined with the function:

Sin(x) = (**Exp**(x)-**Exp**(-x))/2

Example

```
Debug.Show
Trace SinH(0)               // Prints 0
Trace SinH(PI / 2)          // Prints
  2.30129890230729
Trace SinH(PI)              // Prints
  11.5487393572577
Trace SinH(2 * PI)          // Prints
  267.744894041016
Trace SinH(2.14)            // Prints 4.19089...
Trace SinH(ArSinH(2.14)) // Prints 2.14
```

Remarks

SinH() is the reverse function of **ArSinH**().

See Also

CosH(), TanH(), ArSinH(), ArCosH(), ArTanH()

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

CosH Function

Purpose

Returns the hyperbolic cosine of a numeric expression.

Syntax

CosH(x)

Description

The hyperbolic cosine applies to all real numbers greater than or equal to 0. It is obtained with the function:

CosH(x) = (**Exp**(x)+**Exp**(-x))/2

The function y=**CosH**(x) returns in y a real number greater than or equal to 1.

Example

```
Debug.Show
Trace CosH(2.14)         // Prints
  4.30854623595395
Trace CosH(ArCosH(2.14)) // Prints 2.14
```

Remarks

CosH() is the reverse function of **ArCosH**().

See Also

SinH(), TanH(), ArSinH(), ArCosH(), ArTanH()

{Created by Sjouke Hamstra; Last updated: 27/09/2014 by James Gaite}

Add8 Operator and Function

Purpose

Adds a numeric expression to a numeric variable of type **Large**.

Syntax

l = x **Add8** y(operator)

l = **Add8**(i, j[,m,...])(function)

x:Large numeric variable
y:any numeric expression
i, j, m, l:Large integer expression

Description

The operator i **Add8** j and function **Add8**(i, j, ...) return the sum of 64-bit integer expressions. In case one of the parameters isn't a **Large**, it is converted to 64-bit values first (using **CLarge**).

Note There is no 64-bit version of the **Add** command to add an expression to a variable, because **Add** is type independent and works with **Large** types as well.

Example

```
Debug.Show
Dim b# = 1.5, i64 As Large
Trace b# Add8 3          // CInt(b#) + 3 = 5
Trace Add8(b#, 3)        // CInt(b#) + 3 = 5
```

```
Add i64, 3 : Trace i64  // b# = 3
b# = 2.5
Trace b# Add8 3         // CInt(b#) + 3 = 5
Trace Add8(b#, 3)        // CInt(b#) + 3 = 5
```

Remarks

Although **Add** can be used with any numeric variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed.

Instead of **Add** x, y, you can use x = x + y, x := x + y, or x += y. When using integer variables **Add** doesn't test for overflow!

See Also

Add8, Sub8, Mul8, Div8, Mod8

{Created by Sjouke Hamstra; Last updated: 23/09/2014 by James Gaite}

Sub8 Function

Purpose

Subtracts two **Large** integer expressions.

Syntax

large = i **Sub8** j(operator)

large = **Sub8**(i, j [, m, ...])(function)

i, j, m: = Large integer) expression

Description

Sub8 returns the difference between two **Large** integer expressions i and j. The values i and j are converted to 64-bit integer values before the function is applied.

Note There is no 64-bit version of the **Sub** command to add an expression to a variable, because **Sub** is type independent and works with **Large** types as well.

Example

```
OpenW # 1
Print Sub8(5 ^ 3, 4 * 20 + 3)// prints 42
```

Remarks

The **Add8**(), **Sub8**(), **Mul8**() and **Div8**() functions can be mixed freely with each other. For example

```
l% = Sub8(5 ^ 3, 4 * 20 + 3)
// or
l% = Sub8(5 ^ 3, Add8(Mul8(4, 20), 3))
```

See Also

Add8, Sub8, Mul8, Div8, Mod8, +, -, *, /F, \, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 23/10/2014 by James Gaite}

Div8 Function

Purpose

Divides two or more 64-bit integer (Large) expressions.

Syntax

l = x **Div8** y(operator)
l = **Div8(**x, y [,m,...])(function)

x, y, m , l:large exp

Description

Div8 is a **Large** integer division function. Values are converted to 64-bit integers before the division is performed. Internally, GFA-BASIC 32 uses **CLarge**() for the conversion (which isn't the same as **Round**).

Note There is no 64-bit version of the **Div** command to add an expression to a variable, because **Div** is type independent and works with **Large** types as well.

Example

```
' Div as operator
Print 2 Div8 2.50        // Prints 1
Print 2 Div8 2.51        // Prints 0
' Div as a function
Print Div8(126, Succ(2))// Prints 42
```

Remarks

Although **Div8** can be used with any numeric data type variable, the usage of integer variables is recommended in order to achieve the maximum optimization for speed (no coercion to 32 bit before the operation is performed).

Div8 doesn't test for overflow!

See Also

Add8, Sub8, Mul8, Mod8, +, -, *, /F, \, ++, --, +=, -=, /= , *=, Operator Hierarchy

{Created by Sjouke Hamstra; Last updated: 01/10/2014 by James Gaite}

Not Function

Purpose

Performs a logical bit-wise Not.

Syntax

Not i

i:integer expression

Description

Not i inverts the bit pattern i.

Example

```
Debug.Show
Trace Bin$(3, 32)
Trace Bin$(10, 32)
Trace Bin$(Not 3, 32)
Trace Bin$(Not 10, 32)
```

Prints:

```
00000000000000000000000000000011
00000000000000000000000000001010
11111111111111111111111111111100
11111111111111111111111111110101
```

Remarks

~ is synonymous with **Not** and can be used instead.

See Also

And, Or, Xor, Imp, Eqv, %&, |, ~

{Created by Sjouke Hamstra; Last updated: 20/10/2014 by James Gaite}