# Microsoft Visual Basic

## API Reference Contents

## Visual Basic Functions

The VBAPI.LIB file provides access to a subset of the functions residing in VB.EXE, VBRUN100.DLL, VBRUN200.DLL, and VBRUN300.DLL. In Visual C++, custom control support is provided by either MFC200.LIB or MFC200.DLL. These functions include many of the same functions used internally to support standard controls.

The VBAPI.LIB file also provides a set of the Basic language support functions that are useful when creating your own DLL routines. This set of functions supports Basic language features such as arrays and the Variant data type. Typically, you would not use these functions in a custom control, unless you were exporting functions that are called directly from Visual Basic.

Functions that are specific to a version level of Visual Basic are denoted by the version number inside brackets. For example, version 3.0 is denoted by [3.0].

Visual Basic String Manipulation
C String Manipulation
Array Access
Binding Support
Control Access
Control Model
Data Conversion
DDE Management
Error Handling
Event Handling
File Input/Output
Floating-Point State
Messages
Palette Access
Picture Structures
Property Access
Scale Conversions
Status Information
Variant Access
Window Access

## Visual Basic Messages

Messages that are passed to custom controls can be divided into three categories:

1. Model messages (for example VBM_GETDEFSIZE, VBM_HELP) are sent when information about the control's model structure is required. The control handle (HCTL) passed with these messages is always NULL.
2. Instance messages are sent to an actual instance of the control, and the control handle passed with these messages is always valid.
3. Notification messages are sent to a control when it has sent a WM_ message to its parent. These messages have the prefix VBN_.

**Important** Controls must not dereference the control handle when it is passed with a model message.

VBAPI.H defines a number of messages that originate with Visual Basic. These messages are listed below, according to category.

Messages that are specific to a version level of Visual Basic are denoted by the version number inside brackets. For example, version 3.0 is denoted by [3.0].

General
Binding Support
Clipboard
DDE
Design Time Support
Events
File Input/Output
Graphical Control Support
Key Processing
Palette
Property Access

## Visual Basic String Manipulation Functions

The following table describes Visual Basic string manipulation functions.

| Function | Description |
|---|---|
| **VBCreateHlstr** | Allocate string. |
| **VBCreateTempHlstr** [2.0] | Create temporary string. |
| **VBDerefHlstr** | Get pointer to string data. |
| **VBDerefHlstrLen** [2.0] | Get length and pointer to string data. |
| **VBDerefZeroTermHlstr** [2.0] | Get pointer to null-terminated string. |
| **VBDestroyHlstr** | Remove language string. |
| **VBGetHlstr** [2.0] | Copy string to buffer. |
| **VBGetHlstrLen** | Get string length. |
| **VBResizeHlstr** [2.0] | Reallocate string. |
| **VBSetHlstr** | Assign new string data. |

## C String Manipulation Functions

The following table describes Visual Basic C string manipulation functions.

| Function | Description |
|---|---|
| **VBCreateHsz** | Allocate HSZ string. |
| **VBDerefHsz** | Get pointer to string data. |
| **VBDestroyHsz** | Remove string. |
| **VBLockHsz** | Get pointer to data and prevent string from moving. |
| **VBUnlockHsz** | Unlock string address. |

## Binding Support Functions

The following table describes the Visual Basic binding support functions.

| Function | Description |
|---|---|
| **VBGetDataSourceControl** | Retrieve data control currently bound to. |

## Control Access Functions

The following table describes Visual Basic control access functions.

| Function | Description |
|---|---|
| **VBClientToScreen** [2.0] | Convert client points to screen coordinates. |
| **VBGetCapture** [2.0] | Get mouse capture. |
| **VBGetClientRect** [2.0] | Get client coordinates. |
| **VBGetControl** [2.0] | Get specified control. |
| **VBGetControlRect** [2.0] | Get screen coordinates. |
| **VBGetRectInContainer** [2.0] | Get control's container rectangle. |
| **VBInvalidateRect** [2.0] | Add rectangle to update region. |
| **VBIsControlEnabled** [2.0] | Determine user-input state. |
| **VBIsControlVisible** [2.0] | Determine visibility state. |
| **VBMoveControl** [2.0] | Change position and size. |
| **VBReleaseCapture** [2.0] | Release mouse capture. |
| **VBScreenToClient** [2.0] | Convert screen points to client coordinates. |
| **VBSetCapture** [2.0] | Set mouse capture. |
| **VBSetControlFlags** [2.0] | Specify control characteristics. |
| **VBUpdateControl** [2.0] | Update painting of client area. |
| **VBZOrder** [2.0] | Change the Z-order. |

## Control Model Functions

The following table describes Visual Basic control model functions.

| Function | Description |
|---|---|
| **VBGetControlModel** | Get model structure. |
| **VBRegisterModel** | Register a control class. |

## DDE Management Functions

The following table describes Visual Basic DDE management functions.

| Function | Description |
|---|---|
| **VBLinkMakeItemName** [2.0] | Create item name that contains control array information. |
| **VBLinkPostAdvise** [2.0] | Send update notification. |
| **VBPasteLinkOk** [2.0] | Send OK for Paste Link request. |

## Error Handling Functions

The following table describes the Visual Basic error handling functions.

| Function | Description |
|---|---|
| **VBSetErrorMessage** | Set text of next error message. |
| **VBRuntimeError** [2.0] | Generate run-time error. |

## Event Handling Function

The following table describes the Visual Basic event handling function.

| Function | Description |
|---|---|
| **VBFireEvent** | Execute event procedure. |

## File Input/Output Functions

The following table describes Visual Basic file input/output functions.

| Function | Description |
|---|---|
| **VBReadBasicFile** | Read data file. |
| **VBReadFormFile** | Read property value from disk during a load. |
| **VBRelSeekBasicFile** | Move position in data file a relative distance. |
| **VBRelSeekFormFile** | Move position in form file a relative distance. |
| **VBSeekBasicFile** | Move position in data file. |
| **VBSeekFormFile** | Move position in form file. |
| **VBWriteBasicFile** | Write to data file. |
| **VBWriteFormFile** | Write property value to disk during a save. |

## Floating-Point State Functions

The following table describes Visual Basic floating-point state functions.

| Function | Description |
|---|---|
| **VBCbSaveFPState** [2.0] | Save current floating-point state. |
| **VBRestoreFPState** [2.0] | Restore saved floating-point state. |

## Messages Functions

The following table describes Visual Basic messages functions.

| Function | Description |
|---|---|
| **VBDefControlProc** | Default message processing. |
| **VBDerefControl** | Get pointer to programmer-defined structure. |
| **VBSendControlMsg** | Send message to a control. |
| **VBSuperControlProc** | Call superclass directly. |

## Palette Access Functions

The following table describes Visual Basic palette access functions.

| Function | Description |
|---|---|
| **VBAllocPicEx** [2.0] | Allocate HPIC. |
| **VBGetPicEx** [2.0] | Allocate HPIC. |
| **VBPaletteChanged** [2.0] | Get palette change state. |
| **VBTranslateColor** [2.0] | Convert RGB to palette color. |

## Picture Structures Functions

The following table describes Visual Basic picture structures functions.

| Function | Description |
|---|---|
| **VBAllocPic** | Allocate HPIC structure. |
| **VBFreePic** | Decrement reference count and delete HPIC if count is zero. |
| **VBGetPic** | Dereference picture data. |
| **VBPicFromCF** | Get picture from Clipboard. |
| **VBRefPic** | Increment reference count. |

## Property Access Functions

The following table describes Visual Basic property access functions.

| Function | Description |
|---|---|
| **VBDialogBoxParam** | Create a pop-up dialog box. |
| **VBGetControlProperty** | Get property value. |
| **VBSetControlProperty** | Set property value. |

## Scale Conversions Functions

The following table describes Visual Basic scale conversions functions.

| Function | Description |
|---|---|
| **VBXPixelsToTwips** | Convert X units to twips. |
| **VBXTwipsToPixels** | Convert X units to pixels. |
| **VBYPixelsToTwips** | Convert Y units to twips. |
| **VBYTwipsToPixels** | Convert Y units to pixels. |

## Status Information Functions

The following table describes Visual Basic status information functions.

| Function | Description |
|---|---|
| **VBDirtyForm** [2.0] | Indicate property change. |
| **VBGetAppTitle** | Get application title. |
| **VBGetMode** | Determine whether in design, run, or break mode. |
| **VBGetVersion** [2.0] | Get Visual Basic version. |

## Window Access Functions

The following table describes Visual Basic window access functions.

| Function | Description |
|---|---|
| **VBGetControlHwnd** | Get handle to window. |
| **VBGetHInstance** | Get handle to current instance of Visual Basic. |
| **VBGetHwndControl** | Get handle to control. |
| **VBRecreateControlHwnd** | Destroy and recreate window, to enable new window styles. |

## Array Access Functions

The following table describes Visual Basic window access functions.

| Function | Description |
|---|---|
| **VBArrayBounds** [2.0] | Get upper and lower bounds. |
| **VBArrayElement** [2.0] | Get pointer to array element. |
| **VBArrayElemSize** [2.0] | Get size of array element. |
| **VBArrayFirstElem** [2.0] | Get pointer to first array element. |
| **VBArrayIndexCount** [2.0] | Get array indexes. |

## Data Conversion Function

The following table describes Visual Basic data conversion function.
that are new for Visual Basic version 2.0 are marked by [2.0].

| Function | Description |
|---|---|
| **VBFormat** [2.0] | Format data value. |

## Variant Access Functions

The following table describes Visual Basic variant access functions.

| Function | Description |
|---|---|
| **VBCoerceVariant** [2.0] | Convert Variant to data type. |
| **VBGetVariantType** [2.0] | Get Variant data type. |
| **VBGetVariantValue** [2.0] | Get Variant value. |
| **VBSetVariantValue** [2.0] | Set Variant value. |

## Property Access Messages

The following table describes Visual Basic property access messages.

| Message | Description |
|---|---|
| VBM_CHECKPROPERTY | Check property value. |
| VBM_GETPROPERTY | Get property value. |
| VBM_GETPROPERTYHSZ | Get string to display in Properties window. |
| VBM_INITPROPPOPUP | Determine how value is set in Properties window. |
| VBM_SETPROPERTY | Set property value. |

## Events Messages

The following table describes Visual Basic events messages.

| Message | Description |
|---|---|
| VBM_DRAGDROP | Item dropped on control. |
| VBM_DRAGOVER | Item dragged over control. |
| VBM_FIREEVENT | Implement delayed event. |
| VBM_SELECTED   [2.0] | Control selected at design time. |

## Binding Support Messages

The following table describes Visual Basic binding support messages.

| Message | Description |
|---|---|
| VBM_DATA_AVAILABLE [3.0] | Notification of data available from the data control. |
| VBM_DATA_GET [3.0] | Get data from the data control. |
| VBM_DATA_METHOD [3.0] | Send request to data control. |
| VBM_DATA_REQUEST [3.0] | Notification of data requested from the data control. |
| VBM_DATA_SET [3.0] | Send data to the data control. |

## Clipboard Messages

The following table describes Visual Basic Clipboard messages.

| Message | Description |
|---|---|
| VBM_COPY | Copy data from Clipboard. |
| VBM_PASTE | Paste data into Clipboard. |
| VBM_QPASTEOK | Determine if Paste or Paste Link should proceed. |

## File Input/Output Messages

The following table describes Visual Basic file input/output messages.

| Message | Description |
|---|---|
| VBM_LOADPROPERTY | Load property from disk. |
| VBM_LOADTEXTPROPERTY [2.0] | Load properties as text. |
| VBM_SAVEPROPERTY | Save property to disk. |
| VBM_SAVETEXTPROPERTY [2.0] | Save properties as text. |

## Key Processing Messages

The following table describes Visual Basic key processing messages.

| Message | Description |
|---|---|
| VBM_ISMNEMONIC [2.0] | Mnemonic entered. |
| VBM_MNEMONIC | Respond to mnemonic. |
| VBM_WANTSPECIALKEY [2.0] | Virtual key entered. |

## Design Time Support Messages

The following table describes Visual Basic design time support messages.

| Message | Description |
|---|---|
| VBM_GETDEFSIZE   [2.0] | Get default size. |
| VBM_PAINTMULTISEL   [2.0] | Respond to multiple selection. |
| VBM_PAINTOUTLINE   [2.0] | Respond to moving. |

## Graphical Control Support Messages

The following table describes Visual Basic graphical control support messages.

| Message | Description |
|---|---|
| VBM_HITTEST [2.0] | Return mouse hit status. |
| VBM_PAINT [2.0] | Repaint graphical control. |

## Palette Messages

The following table describes Visual Basic palette messages.

| Message | Description |
|---|---|
| VBM_GETPALETTE [2.0] | Request for palette. |
| VBM_PALETTECHANGED [2.0] | Request to select a palette. |

## DDE Messages

The following table describes Visual Basic DDE messages.

| Message | Description |
|---|---|
| VBM_LINKENUMFORMATS   [2.0] | Enumerate DDE data formats. |
| VBM_LINKGETDATA   [2.0] | Receive DDE data. |
| VBM_LINKGETITEMNAME   [2.0] | Get DDE item name. |
| VBM_LINKSETDATA   [2.0] | Send DDE data. |

## General Messages

The following table describes Visual Basic general messages.

| Message | Description |
|---|---|
| VBM_CANCELMODE | Reset internal state. |
| VBM_CREATED | Creation and loading of a control completed. |
| VBM_HELP | Process help request. |
| VBM_INITIALIZE | HCTL allocated. |
| VBM_LOADED | All controls on form loaded, or control dynamically loaded. |
| VBM_METHOD | One of control's methods used in a statement. |

## Technical Notes

You can click on any file to launch Notepad and load that file.

TN001.TXT−"Support for DT_OBJECT Properties"

TN002.TXT−"Custom Control Version Management"

## Windows Process Status utility

### Description

The Windows Process Status utility, WPS.EXE, displays a listing of all running tasks along with a listing of all modules that are currently loaded into memory. This utility is useful for monitoring the memory usage of an application with regard to its modules.

In addition, the WPS.EXE utility can be used to force a task to terminate, or to unload a module that remained in memory due to an application error.

## Other Information Sources

All help and text files included with the Standard Edition are also included with the Professional Edition.

**Standard Edition**

Help Files
Text Files

**Professional Edition**

Help Files
Text Files

---

**Note** When Visual Basic is installed, as each Help file is installed, it is listed in WINHELP.INI, located in your Windows directory. If you move a Help file to a different directory, be sure to change the path in WINHELP.INI. If a Help file does not exist, is not in your path, or is not in the directory specified in WINHELP.INI, WinHelp displays an appropriate message.

---

## Standard Edition Help Files

You can click on any help file to go to the main table of contents of that file. If the file is not available, an error occurs.

Visual Basic Help−Documents Visual Basic for Windows.

SetupWizard−Documents the SetupWizard application. For information about the Setup Toolkit, search for Setup in the Visual Basic help file.

Data Manager−Documents the Data Manager application.

---

**Note** When Visual Basic is installed, as each Help file is installed, it is listed in WINHELP.INI, located in your Windows directory. If you move a Help file to a different directory, be sure to change the path in WINHELP.INI. If a Help file does not exist, is not in your path, or is not in the directory specified in WINHELP.INI, WinHelp displays an appropriate message.

---

## Professional Edition Help Files

In addition to the help files provided with the Standard Edition, these help files are included with the Professional Edition. You can click on any help file to go to the main table of contents of that file. If the file is not available, an error occurs.

Crystal Reports—Documents the Crystal Reports application.

Custom Control Reference—Documents each of the cusom controls provided with the Professional Edition.
Help Compiler Reference—Documents the Windows Help application for Help writers and programmers.
Hotspot Editor—Documents the segmented hypergraphic editor for creating hotspots within graphics for use in authoring Help files.
KnowledgeBase—A collection of articles from Microsoft Technical Support with tips, ideas and solutions.
ODBC Installation Help—Documents the installation tools for ODBC.

Oracle ODBC Driver—Documents the ODBC driver for Oracle databases.
SQL Server ODBC Driver—Documents the ODBC driver for SQL Server databases.
Visual Basic API Reference—Documents the Custom Control Development Kit.
Windows 3.1 API for Visual Basic—Declarations, structures, and constants for the Windows API as used in Visual Basic.
Windows 3.1 SDK Help—Documents Windows functions as used in the C programming language.

**Note** When Visual Basic is installed, as each Help file is installed, it is listed in WINHELP.INI, located in your Windows directory. If you move a Help file to a different directory, be sure to change the path in WINHELP.INI. If a Help file does not exist, is not in your path, or is not in the directory specified in WINHELP.INI, WinHelp displays an appropriate message.

## Standard Edition Text Files

You can click on any of these files to launch Notepad and load the file.

README.TXT−Information on last minute changes to Visual Basic, as well as additional information.

CONSTANT.TXT−Global symbolic constants for Visual Basic properties, events, functions and statements.
DATACONS.TXT−Global symbolic constants for the data access features of Visual Basic.
EXTERNAL.TXT−Additional README information about connecting to external databases.
PACKING.LST−List of all files on the distribution disks provided with Visual Basic.

## Professional Edition Text Files

In addition to the text files provided with the Standard Edition, these text files are included with the Professional Edition.   You can click on any file to launch Notepad and load that file. If the file is too large for Notepad, you may have to use a different word processor.

BTRIEVE.TXT−Supplementary information on importing, exporting, or attaching Btrieve tables with Visual Basic.

ORACLE.TXT−Setup information for the ODBC Oracle driver to run with your ORACLE RDBMS software.   If you installed ODBC, this file is in your Windows\SYSTEM directory.

PERFORM.TXT−Performance tuning tips for Visual Basic version 3.0 and Microsoft Access (TM) Relational Database System for Windows version 1.1.

SAMPLES.TXT−List of applications written in Visual Basic that demonstrate techniques discussed in the printed documentation.

WIN30API.TXT−Global symbolic constants for Windows 3.0 API functions.

WIN31API.TXT−Global symbolic constants for Windows 3.1 API functions.

WINMMSYS.TXT−Type declarations and global symbolic constants for Windows 3.1 multimedia API functions.

## VBAllocPic

See Also    Example

### Syntax

**HPIC VBAllocPic**(*lppic*)

Allocates an internal picture structure from information supplied in a PIC structure. Once allocated, information about the picture can be accessed by calls to **VBGetPic**, which supplies a pointer to a PIC structure.

**Note** Given a choice, the **VBAllocPicEx** function is preferred over the **VBAllocPic** function.

| Parameter | Type | Description |
|---|---|---|
| *lppic* | **LPPIC** | Far pointer to a PIC structure, which is filled with valid data for a bitmap, icon, or metafile. |

### Comments

Use of this function is generally necessary only when you use custom processing for handling a DT_PICTURE property. A DT_PICTURE property should be represented in the programmer-defined structure as an HPIC data type. A pointer to an HPIC handle should be supplied when responding to VBM_GETPROPERTY.

If the picture is to be retained, you should call **VBRefPic** to increment the reference count from zero. If the reference count is left at zero, Visual Basic deletes the picture as soon as the control procedure returns. The reference count does not need to be incremented if the HPIC handle is a temporary handle supplied to support VBM_GETPROPERTY.

Once allocated, the bitmap, icon, or metafile should not be deleted or modified.

### Return Value

An HPIC handle to a picture structure.

**See Also**

VBAllocPicEx
VBFreePic
VBGetPic
VBPicFromCF
VBRefPic

**Example**

```
// Create HPIC from HBMP.
pic.picType = PICTYPE_BITMAP;
pic.picData.bmp.hbitmap = hBmp;
hpic = VBAllocPic(&pic);
VBRefPic(hpic);        // Increment reference count
```

## VBAllocPicEx [2.0]

See Also Example

### Syntax

**HPIC VBAllocPicEx**(*lpPic*, *usVersion*)

This function is almost identical to the **VBAllocPic** function. The only difference is that the PIC data structure in Visual Basic version 2.0 contains an *hpal* member. Refer to **VBAllocPic** for more information.

| Parameter | Type | Description |
|---|---|---|
| *lpPic* | **LPPIC** | Far pointer to PIC structure. |
| *usVersion* | **USHORT** | The Visual Basic version. |

### Comments

If *usVersion* is greater than or equal to VB200_VERSION, then the *hpal* member can be used in the PIC structure.

### Return Value

The allocated HPIC handle, or NULL if an error occurs.

**See Also**

VBAllocPic
VBFreePic
VBGetPic
VBPicFromCF
VBRefPic

**Example**

```
// Create HPIC from PIC.
pic.picType = PICTYPE_BITMAP;
pic.picData.bmp.hbitmap = hBmp;
pic.picData.bmp.hpal = hPal;
hpic = VBAllocPicEx(&pic, VB_VERSION);
```

## VBArrayBounds [2.0]

See Also Example

### Syntax

**LONG VBArrayBounds**(*hAD*, *index*)

Returns a long integer that contains the lower and upper bounds of a Basic language array.

| Parameter | Type | Description |
|---|---|---|
| *hAD* | **HAD** | Handle to array descriptor. |
| *index* | **SHORT** | The array dimension 1 is the first dimension, 2 is the second dimension for a two dimensional array, and so on. |

### Return Value

The low word of the return value is the lower bound of the array dimension; the high word is the upper bound. If *index* is not valid or *hAD* refers to a dynamic array that has been reinitialized (**Erase** statement) or never allocated (**ReDim** statement), AB_INVALIDINDEX is returned. The *hAD* value must be passed from Visual Basic as a parameter that is declared as an array.

**See Also**

**VBArrayIndexCount**

**Example**

```
// Get all the dimensions of an array.
index = -1;

do {
  index++;
  lBounds = VBArrayBounds(hAD, index + 1);
  usLow[index] = LOBOUND(lBounds);
  usHigh[index] = HIBOUND(lBounds);
} while (lBounds != AB_INVALIDINDEX);

// The index variable contains number of indexes in array.
if (index == 0)
// Dynamic array not allocated.
```

## VBArrayElement [2.0]

See Also Example

### Syntax

**LPVOID VBArrayElement**(*hAD*, *cIndex*, *lpi*)

Returns a pointer to the value of a Basic language array element.

| Parameter | Type | Description |
|---|---|---|
| *hAD* | **HAD** | Handle to array descriptor. |
| *cIndex* | **SHORT** | Number of array indexes. |
| *1pi* | **LPSHORT** | Pointer to an array of indexes. |

### Comments

If *cIndex* does not match the actual number of indexes for the array, or any of the indexes are out of bounds, then the high word (segment) of the return value will be zero, and the low word (offset) will contain the error code.

### Return Value

Returns a pointer to the value of the array element. If the array is a string array, the function returns an HLSTR value.The *hAD* value must be passed from Visual Basic as a parameter that is declared as an array.

**See Also**

VBArrayElemSize
VBArrayIndexCount

### Example

```
SHORT indexes[2];

indexes[0] = 2;
indexes[1] = 2;

// Get the (2, 2) item of a 2-dimensional array.
lpData = VBArrayElement(hAD, 2, indexes);
if (HIWORD(lpData) == 0)
  return (ERR)LOWORD(lpData);   // Return error.
```

## VBArrayElemSize [2.0]

See Also

### Syntax

**USHORT VBArrayElemSize**(*hAD*)

Returns the size of a single element in a Basic language array.

| Parameter | Type | Description |
|---|---|---|
| *hAD* | **HAD** | Handle to array descriptor. |

### Comment

Can be used with the **VBArrayFirstElem** function to perform manual array indexing; adding multiples of the array element size to the offset of the first element in the array results in the address of the *nth* element.

This function is not meaningful for arrays of variable-length strings.

### Return Value

Returns the size of a single element in the array. The *hAD* value must be passed from Visual Basic as a parameter that is declared as an array.

**See Also**

VBArrayElement
VBArrayFirstElem

## VBArrayFirstElem   [2.0]

See Also

### Syntax

**LPVOID VBArrayFirstElem**(*hAD*)

Returns a pointer to the first element of a Basic language array.

| Parameter | Type | Description |
|---|---|---|
| *hAD* | **HAD** | Handle to array descriptor. |

### Comments

This function is not meaningful for arrays of variable-length strings.

### Return Value

Returns a pointer to the first element of the array, except for string arrays, which have no meaningful return value. The *hAD* value must be passed from Visual Basic as a parameter that is declared as an array. If *hAD* refers to a dynamic array that has been reinitialized (**Erase** statement) or never allocated (**ReDim** statement), NULL is returned.

**See Also**

VBArrayElement
VBArrayElemSize

## VBArrayIndexCount [2.0]

See Also Example

### Syntax

**SHORT VBArrayIndexCount**(*hAD*)

Returns the number of indexes for a Basic language array.

| **Parameter** | **Type** | **Description** |
|---|---|---|
| *hAD* | **HAD** | Handle to array descriptor. |

### Return Value

Returns a count of the number of indexes for the array. The *hAD* value must be passed from Visual Basic as a parameter that is declared as an array. If *hAD* refers to a dynamic array that has been reinitialized (**Erase** statement) or never allocated (**ReDim** statement), 0 is returned.

**See Also**

**VBArrayBounds**

**Example**

```
if ((cIndex = VBArrayIndexCount(hAD)) == 0)
    return 0;   // Return error
```

## VBCbSaveFPState [2.0]

See Also

### Syntax

**USHORT VBCbSaveFPState**(*lpBuff*, *cb*)

Saves the current floating-point state.

| Parameter | Type | Description |
|---|---|---|
| *lpBuff* | **LPVOID** | A far pointer to a buffer that receives the current floating-point state. |
| *cb* | **USHORT** | The size of *lpBuff*, or 0 if you want to return the size required to hold the current floating-point state. |

### Comments

Saves the current floating-point state, regardless of whether a floating-point coprocessor or emulator is used. Use the **VBRestoreFPState** function to restore the saved floating-point state.

The **VBCbSaveFPState** and **VBRestoreFPState** functions should only be called from custom controls that implement the VBM_GETPROPERTY message if your code for processing this message causes your control to yield to another application. Examples of actions that yield control to other applications are:

- Displaying Windows message boxes
- Displaying dialog boxes
- Executing a DDE action

When Visual Basic sends a VBM_GETPROPERTY message to a custom control, intermediate floating-point computations may be on the floating-point stack. All applications on the system share the same floating-point stack. If the code in the custom control handles one of the messages and yields to another application, the intermediate floating-point computations on the stack may be destroyed.

### Return Value

Returns 0 if the floating-point state is successfully copied to *lpBuff*. Otherwise, the function returns the number of bytes needed to store the floating-point state.

**See Also**

VBRestoreFPState

## VBClientToScreen [2.0]

See Also

### Syntax

**VOID VBClientToScreen**(*hctl*, *lpPt*)

Converts the given point from client coordinates to screen coordinates. Screen coordinates are relative to the upper-left corner of the screen.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *lpPt* | **LPPOINT** | Points to a **POINT** data structure that contains the coordinates to be converted. |

### Comments

This function is similar to the Windows API **ClientToScreen** function, except that the **VBClientToScreen** function can be used by graphical controls, whereas the **ClientToScreen** function can not.

### Return Value

None.

**See Also**

ClientToScreen (in Windows SDK)
VBScreenToClient

## VBCoerceVariant [2.0]

See Also Example

### Syntax

**ERR VBCoerceVariant**(*lpVar*, *vtype*, *lpData*)

Copies the value of a Variant to a memory location, coercing it to the specified data type.

| Parameter | Type | Description |
|---|---|---|
| *lpVar* | **LPVAR** | Far pointer to Variant. |
| *vtype* | **SHORT** | The desired data type of the Variant data value. |
| *lpData* | **LPVOID** | Far pointer to data buffer. Buffer must be large enough to store value of coerced Variant. |

### Comments

Refer to the **VBGetVariantType** function for a list of Variant data types.

If this function returns a string (HLSTR), it may or may not be a temporary string. Eventually, the string must be freed using a Visual Basic function that frees strings, such as **VBGetHlstr**.

### Return Value

If *vtype* is invalid, the function returns 1. For any other errors, the function returns ERR. A return value of 0 indicates a successful coersion to the given type.

**See Also**

**VBGetHlstr**
**VBGetVariantType**
**VBGetVariantValue**

## Example

```
ERR err;
LONG i4;

// Coerce Variant to LONG.
err = VBCoerceVariant(lpVar, VT_I4, &i4)
  return err;   // Return ERR value
```

## VBCreateHlstr

See Also Example

### Syntax

**HLSTR VBCreateHlstr**(*pb*, *cbLen*)

Gets a Basic language string from the given data. A Basic language string (HLSTR) is managed as part of the Visual Basic string space; the Visual Basic user can treat it as an ordinary string. You must call this function before including a string as a parameter in a call to **VBFireEvent**.

| Parameter | Type | Description |
|---|---|---|
| *pb* | **LPVOID** | Far pointer to buffer containing the string data. If NULL, the string is uninitialized. |
| *cbLen* | **USHORT** | The number of bytes in the string. If set to zero, the *pb* parameter is not used. |

### Comments

The string (HLSTR) must eventually be freed with **VBDestroyHlstr**.

### Return Value

A handle to a Basic language string, or NULL if Visual Basic runs out of memory.

**See Also**

**VBDerefHlstr**
**VBDestroyHlstr**
**VBGetHlstrLen**
**VBSetHlstr**

**Example**

```
cbCaption = GetWindowText(hwnd, pStrBuf, 20);
params.ClickString = VBCreateHlstr(pStrBuf, cbCaption);
```

## VBCreateHsz

See Also Example

### Syntax

**HSZ VBCreateHsz(***segment*, *lpszString***)**

Dynamically allocates a null-terminated string that may be moved around in memory. Visual Basic keeps track of the string indirectly, through use of a handle, because the actual address of the data may change. A custom property that contains a string is declared as an HSZ type, and the corresponding field in the programmer-defined structure should also be an HSZ type.

| Parameter | Type | Description |
|---|---|---|
| *segment* | **HANDLE** | The segment in which to allocate the string. |
| *lpszString* | **LPSTR** | A pointer to the zero-terminated data to copy into the string. |

### Comments

When a string property is set, Visual Basic either assigns a new value directly to the HSZ field (if the PF_fSetData flag is on) or sends a VBM_SETPROPERTY message. If you set the property yourself, you should call **VBDestroyHsz** to free the current string data and then call **VBCreateHsz** again to allocate space for the new data.

You can dereference the handle (thereby getting a pointer to the string data) by calling either function **VBDerefHsz** or **VBLockHsz**.

The string can be allocated in any segment initialized as a Windows local heap. The segment can also be the same as that of the control structure (*hctl*), which indicates that the controls heap should be used.

### Return Value

A handle to the new string, or NULL if memory was not available in the segment requested.

**See Also**

**VBDestroyHsz**
**VBDerefHsz**
**VBLockHsz**
**VBUnlockHsz**

## Example

```
 hsz = CreateHsz((_segment) hctl, "Visual Basic control.");
```

## VBCreateTempHlstr [2.0]

See Also Example

### Syntax

**HLSTR VBCreateTempHlstr**(*pb*, *cbLen*)

Creates a temporary Basic language string from the given data. Temporary strings are automatically deleted the first time they are used by Visual Basic. The string space in Visual Basic allows a maximum of 20 temporary strings.

| Parameter | Type | Description |
|---|---|---|
| *pb* | **LPVOID** | Far pointer to buffer containing the string data. If NULL, the string is uninitialized. |
| *cbLen* | **USHORT** | The number of bytes in *pb*. If set to zero, *pb* is not referenced, and a NULL string is returned. |

### Comments

Temporary strings are used when you are returning a string to Visual Basic directly as a function return value or as a placeholder for the intermediate results of an expression.

**Important** **VBDestroyHlstr** must not be called to free a temporary string. Use a Visual Basic function that deletes temporary strings, such as **VBGetHlstr**.

### Return Value

A handle to a Basic language string. The function returns NULL for a zero-length string. If the high word of the HLSTR is 1, then the function was not successful, and the low word contains the error code.

**See Also**

**VBCreateHlstr**

**VBGetHlstr**

**Example**

```
hlstr = VBCreateTempHlstr(lpzBuffer, lstrlen(lpBuffer));

if (HIWORD(hlstr) == -1)
  return LOWORD(hlstr);    // Return error
```

## VBDefControlProc

See Also Example

### Syntax

**LONG VBDefControlProc**(*hctl*, *hwnd*, *msg*, *wp*, *lp*)

Performs default message processing as required for Visual Basic controls; this function is analagous to the general Windows default processor, **DefWindowProc**.

Its important for control procedures to call this function for default processing. In particular, if the control procedure gets a message but does not process the message itself, it must allow the control to fall through to a call to **VBDefControlProc**. This function performs all processing for messages related to standard properties, as well as for other messages sent by Visual Basic and by Windows.

For a description of all the default processing implemented by **VBDefControlProc**, see the descriptions of default action for the individual VBM messages.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *hwnd* | **HWND** | The handle to the controls window. |
| *msg* | **USHORT** | Integer identifying the message to process. |
| *wp* | **USHORT** | Word parameter of the message. |
| *lp* | **LONG** | Long (32-bit) parameter of the message. |

### Comments

The default control procedure will either process a message completely or pass it along. If there is a superclass, the window procedure of the superclass gets the message. Otherwise, the message is sent to the default window procedure.

### Return Value

A long integer. The meaning depends on the message which is being processed: see descriptions of return value for individual messages.

**See Also**

**DefWindowProc** (in Windows SDK)

**VBSuperControlProc**

**Example**

```
.
. // Message pre-processing
.
lResult = VBDefControlProc(hctrl, hwnd, msg, wp, lp);
.
. // Message post-processing
.
return lResult;
```

## VBDerefControl

See Also Example

### Syntax

**LPVOID VBDerefControl(***hctl*)

Takes a handle to a control structure, and returns a far pointer to the controls *programmer-defined structure*, which is the portion of the control structure to which the custom-control writer has access.

As part of the control structure, the programmer-defined structure is allocated once for each instance of the control. You can use this structure to store information private to each instance, including the value of custom properties. In the control model, you indicate the size of the programmer-defined structure.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control structure. |

### Comments

Whenever you call a Visual Basic API function, you should call **VBDerefControl** again if you need to refer to the programmer-defined structure. Causing an allocation in the control structures own segment is likely to invalidate this pointer, and many Visual Basic API functions can cause such an allocation.

### Return Value

A far pointer to the programmer-defined structure. This pointer is of type LPVOID, and needs to be recast to a pointer to the structure youve defined.

**See Also**

VBGetControlHwnd

VBGetControlModel

**Example**

```
PCIRCLE   pCircle;

if (hctrl)
  pCircle = (PCIRCLE)VBDerefControl(hctrl);
```

## VBDerefHlstr

See Also Example

### Syntax

**LPSTR VBDerefHlstr**(*hlstr*)

Returns a pointer to the string data of a Basic language string. (See **VBCreateHlstr** for more information on this data type.) The pointer returned becomes invalid as soon as the string is moved in memory; any call to a Visual Basic API function may have that effect. If this happens, the string must be dereferenced again to be valid.

The length of the string cannot be changed directly. To assign new data to the string or to change its length, use **VBSetHlstr**. However, **VBDerefHlstr** is useful for examining string contents.

The string data is not null-terminated. Use the **VBGetHlstrLen** function in conjunction with **VBDerefHlstr** to determine the length of the string.

| Parameter | Type | Description |
|---|---|---|
| *hlstr* | **HLSTR** | Handle to the Basic language string. |

### Comments

This function does not free temporary strings.

### Return Value

A far pointer to the string data, or NULL if the string is of zero length. If the string has a length greater than zero, it is not null-terminated and can contain embedded nulls.

**See Also**

**VBCreateHlstr**
**VBDerefHlstrLen**
**VBDerefZeroTermHlstr**
**VBDestroyHlstr**
**VBGetHlstr**
**VBGetHlstrLen**
**VBSetHlstr**

## Example

```
lpstr = VBDerefHlstr(params.ClickString);
```

## VBDerefHlstrLen [2.0]

See Also Example

### Syntax

**LPSTR VBDerefHlstrLen**(*hlstr*, *pcbLen*)

Returns a pointer to the string data of a Basic language string and the length of the string. The returned pointer value becomes invalid as soon as the string is moved in memory; any call to a Visual Basic API function may have that effect. If this happens, the string must be dereferenced again to be a valid pointer.

| Parameter | Type | Description |
|---|---|---|
| *hlstr* | **HLSTR** | Handle to the Basic language string. |
| *pcbLen* | **USHORT FAR *** | A far pointer to an unsigned integer that contains the length of the returned string data. |

### Comments

The function is identical to **VBDerefHlstr**, with the exception that **VBDerefHlstrLen** additionally returns the length of the string data as a parameter.

This function does not free temporary strings.

### Return Value

A far pointer to the string data, or NULL if the string is zero length. If the string has a length greater than zero, it is not null-terminated and can contain embedded nulls.

**See Also**

**VBDerefHlstr**
**VBDerefZeroTermHlstr**
**VBGetHlstr**

### Example

```
lpstr = VBDerefHlstr(params.ClickString, &usLen);
```

## VBDerefHsz

See Also Example

### Syntax

**LPSTR VBDerefHsz**(*hsz*)

Returns a pointer to null-terminated string data. The input to the function is an HSZ type, which is a handle to a null-terminated string managed by Visual Basic. This handle remains valid even as the string is moved around in memory. However, to manipulate the string itself, you must first dereference the handle by calling this function.

The pointer returned becomes invalid as soon as the string is moved in memory; any call to a Visual Basic API function may have that effect. To ensure that the string is not moved, use **VBLockHsz** instead.

| Parameter | Type | Description |
|---|---|---|
| *hsz* | **HSZ** | Handle to the string. |

### Return Value

A far pointer to the null-terminated string data.

**See Also**

**VBCreateHsz**
**VBDestroyHsz**
**VBLockHsz**
**VBUnlockHsz**

## Example

```
lpstr = VBDerefHsz(hsz);
```

## VBDerefZeroTermHlstr [2.0]

See Also Example

### Syntax

**LPSTR VBDerefZeroTermHlstr**(*hlstr*)

Returns a pointer to the string data; however, the string is null-terminated.

| Parameter | Type | Description |
|---|---|---|
| *hlstr* | **HLSTR** | Handle to the Basic language string. |

### Comments

The function is identical to **VBDerefHlstr**, with the exception that **VBDerefZeroTermHlstr** returns a null-terminated string. Note that Basic strings can have embedded null characters, so there may be null characters within the string in addition to the one at the end.

The pointer returned by this function becomes invalid as soon as the string is moved in memory. You may use **VBDerefHlstr** to dereference the string again.

The null termination character is added to the string in Basics string space, and will remain until a new value is assigned to the string (either in Visual Basic code or with **VBSetHlstr**). The addition of the null termination character does not affect the length of the string, according to **VBGetHlstrLen** or the Basic **Len** function.

### Return Value

A far pointer to the null-terminated string data, or NULL if an error occurs.

**See Also**

VBDerefHlstr

VBDerefHlstrLen

## Example

```
lpszData = VBDerefZeroTermHlstr(params.ClickString);
```

## VBDestroyHlstr

See Also Example

### Syntax

**VOID VBDestroyHlstr**(*hlstr*)

Frees memory allocated for a Basic language string. After this function is called, the handle no longer references a valid string.

| Parameter | Type | Description |
|---|---|---|
| *hlstr* | **HLSTR** | Handle to the Basic language string. |

### Comments

This function must not be used with temporary strings. Temporary strings are returned by **VBCreateTempHlstr** and possibly **VBCoerceVariant** and **VBGetVariantValue**. Use **VBGetHlstr** to free temporary strings.

**See Also**

VBCreateHlstr
VBGetHlstr

### Example

```
VBDestroyHlstr(hlstr);
```

## VBDestroyHsz

See Also          Example

### Syntax

**VOID  VBDestroyHsz**(*hsz*)

Frees memory allocated for an HSZ string, which is a null-terminated string allocated by **VBCreateHsz** and managed by Visual Basic. After this function is called, the handle no longer references a valid string.

| Parameter | Type | Description |
|---|---|---|
| *hsz* | **HSZ** | Handle to the string. |

**See Also**

**VBCreateHsz**

### Example

```
VBDestroyHsz(hsz);
```

## VBDialogBoxParam

See Also Example

### Syntax

**int VBDialogBoxParam**(*hinst*, *pszTemplateName*, *lpDialogFunc*, *lp*)

Creates a task-modal dialog box. Being *task-modal* means that the application developer can switch to other applications, but cannot switch to other windows within Visual Basic without first closing the window.

Use this function in place of the Windows **DialogBox** or **DialogBoxParam** function to display a task-modal dialog box. For example, you might want to display a dialog box as the mechanism for setting a property in the Properties window. In response to the VBM_INITPROPPOPUP message, you create a window. When sent a WM_SHOWWINDOW message, the window posts a message to itself that hides itself and creates a dialog box. (See the sample source code in the CIRC3 directory for clarification.)

| Parameter | Type | Description |
|---|---|---|
| *hinst* | **HANDLE** | Handle to the DLL module. This should be the module handle that was saved in **LibMain**. |
| *pszTemplateName* | **LPSTR** | Pointer to a null-terminated string that names the dialog box template. |
| *lpDialogFunc* | **FARPROC** | Pointer to the dialog procedure. |
| *lp* | **LONG** | The long parameter (*lParam*) to be passed to the dialog procedure as part of the WM_INITDIALOG message. |

**See Also**

**DialogBox** (in Windows SDK)

**DialogBoxParam** (in Windows SDK)

VBM_INITPROPPOPUP

**Example**

```
VBDialogBoxParam(hmodDLL, "FlashDlg", DlgProc, 0L);
```

## VBDirtyForm [2.0]

See Also Example

### Syntax

**VOID VBDirtyForm**(*hctl*)

Updates property values before switching from design mode to run mode within Visual Basic. This function should be called whenever you alter property data without calling **VBSetControlProperty** or receiving a VBM_SETPROPERTY message. If **VBSetControlProperty** has been called, or if you change a standard property by calling a Windows function such as **SetWindowText**, Visual Basic already knows that a property value has been changed.

Updates property values before switching from design mode to run mode within Visual Basic.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control structure. |

**See Also**

VBSetControlProperty

**Example**

```
pCircle->FlashColor = tmpColor;
VBDirtyForm(hctl);
```

## VBFireEvent

See Also Example

### Syntax

**ERR VBFireEvent(***hctl*, *iEvent*, *lpparams***)**

Fires the specified event by executing the corresponding event procedure, if it exists. This function waits for the event procedure to finish executing and then returns.

If execution is broken in the middle of the event procedure for any reason, **VBFireEvent** does not return until the user gives a debugging command to finish execution of the event procedure. Because of the possibility of yielding, **VBFireEvent** should not be called in response to certain messages, such as WM_SETFOCUS and WM_KILLFOCUS.

Care should be taken in performing actions after **VBFireEvent** returns. During execution of the Visual Basic event procedure, a great many things can happen: the control, its form, and the segment that contains them may all be removed as a result of **Unload** statements. If possible, calling **VBFireEvent** should be the last thing that the control procedure does before returning. If you need to perform actions after calling **VBFireEvent**, check to insure that the controls window handle is still valid (see Example) before referencing the control structure.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control structure. |
| *iEvent* | **USHORT** | Index to the event within the controls event information table. The first event declared is indexed as 0, the second as 1, and so on. |
| *lpparams* | **LPVOID** | A pointer to the argument structure, which contains a far pointer to each argument in the reverse order in which the arguments appear. If an argument is a string, the structure contains an HLSTR field (not a pointer). The structure also contains an extra pointer to the *Index* argument, which Visual Basic may or may not use. Since the *Index* argument appears first in the event procedure, if at all, it is placed last in the structure.<br>This parameter can be NULL if there are no arguments to the event procedure. |

### Comments

Your code should allocate space for each pointer in the argument list, and make each pointer point to meaningful data (except in the case of the *Index* argument, for which you only need to allocate space for the pointer). If the event procedure changes the value of an argument, the data referenced by the far pointer will be updated.

A string must be translated into a Basic language string before being passed to an event procedure. **VBCreateHlstr** can be used to create a Basic string. The pointer in the argument list should then point to the HLSTR handle returned by **VBCreateHlstr**, **VBGetHlstrLen**, and **VBDerefHlstr** can be used to retrieve the value of the string after the event procedure finishes.

### Return Value

Zero, if no error. Nonzero indicates an error; the control may have been deleted by Visual Basic statements during event procedure execution.

**Note** If an error occurs, you should assume that the control has been destroyed. Therefore, you should return from the control procedure as soon as possible.

**See Also**

**VBCreateHlstr**
**VBDerefHlstr**
**VBDestroyHlstr**
VBM_FIREEVENT
**VBGetHlstrLen**

**Example**

```
typedef struct tagPARAMS
{
  HLSTR hlstrClick;
  LPVOID  Index
}
PARAMS;
.
.
.
PARAMS params;
char StrBuf[20];
int cbCaption, err;
LPSTR lpstr;

cbCaption = GetWindowText(hwnd, StrBuf, 20);
params.hlstrClick = VBCreateHlstr(StrBuf, cbCaption);
err = VBFireEvent(hctrl, EVENT_PUSH_CLICK, &params);
if (!err)
{
  /* Control is still valid; returned args can be processed.*/
}
```

## VBFormat [2.0]

See Also Example

### Syntax

**SHORT VBFormat**(*vtype*, *lpData*, *lpszFmt*, *pb*, *cb*)

Converts a value to a string and formats it according to instructions contained in a format expression. Refer to the Visual Basic **Format$** function for a complete description of format expressions.

| Parameter | Type | Description |
|---|---|---|
| *vtype* | **SHORT** | The data type of the value. The Variant data values are used (VT_*xxx* ). |
| *lpData* | **LPVOID** | The value to be formatted. |
| *lpszFmt* | **LPSTR** | A format expression; must be a null-terminated string. |
| *pb* | **LPVOID** | Buffer to contain the formatted value. |
| *cb* | **USHORT** | Size of *lpBuff*. |

### Comments

Refer to **VBGetVariantType** for a list of all the Variant data types that can be used for the *vtype* parameter.

If *vtype* is set to VT_STRING, *lpData* must be a pointer to an HLSTR. If *lpData* is a pointer to a temporary HLSTR, the HLSTR is deleted after being referenced.

### Return Value

The number of bytes copied to *pb*. If the return value is 1, an error occurred.

**See Also**

VBGetVariantType

**Example**

```
char formatted[20];
int i = 6;

VBFormat(VT_I2, &i, ##, formatted, sizeof(formatted));
```

## VBFreePic

See Also Example

### Syntax

**VOID VBFreePic**(*hpic*)

Decrements the reference count for the picture and deletes the picture structure from memory if the reference count becomes or was already zero. The reference count is the number of different properties that use the same image, so **VBRefPic** or **VBFreePic** should be called each time another property uses or deletes the image.

If the internal picture structure is deleted from memory (because the reference count becomes zero), references to the HPIC handle become invalid.

| Parameter | Type | Description |
|---|---|---|
| *hpic* | **HPIC** | Handle to an internal picture structure. |

### Return Value

None.

**See Also**

- VBAllocPic
- VBGetPic
- VBPicFromCF
- VBRefPic

**Example**

```
case WM_DESTROY:
  ppix = (PPIX)VBDerefControl(hctl);
  VBFreePic(ppix->hpicPicture);
  break;
```

## VBGetAppTitle

Example

### Syntax

**VOID VBGetAppTitle**(*lpstr*, *cbMax*)

Gets a string containing the title of the current application. This string, which can contain embedded spaces, is entered by the application developer when using the Make EXE File command. The application title is displayed by the Windows Task List and by the Program Manager.

The **VBGetAppTitle** function copies a null-terminated string to the specified location, but will in any case not copy more bytes than specified in the *cbMax* parameter.

| Parameter | Type | Description |
|---|---|---|
| *lpstr* | **LPSTR** | Location to which to copy the application title. |
| *cbMax* | **USHORT** | Maximum number of bytes to copy. |

#### Example

```
char   szBuf[21];

VBGetAppTitle((LPSTR)szBuf, 20);
```

## VBGetCapture [2.0]

See Also Example

### Syntax

**HCTL VBGetCapture**()

Retrieves a handle that identifies the control that has the mouse capture. *Mouse capture* is the ability to receive all mouse input and to block other objects from receiving mouse input. Only one control can have the mouse capture at any given time. If a control does have the mouse captured, the control receives mouse input whether or not the cursor is within its borders.

### Return Value

The return value identifies the control that has the mouse capture; it is NULL if no control has the mouse capture.

---

**Note** A NULL return value does not indicate that no window has the mouse captured, only that if a window does, it does not belong to a control.

---

**See Also**

**GetCapture** (in Windows SDK)

### Example

```
hctl = VBGetCapture();
```

## VBGetClientRect [2.0]

See Also Example

### Syntax

**VOID VBGetClientRect**(*hctl*, *lpRect*)

Copies the dimensions of the client area of a control into the structure pointed to by the *lpRect* parameter. Since the returned client **RECT** is in client coordinates (relative to the upper-left corner of a controls client area), the coordinates of the upper-left corner are (0, 0).

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *lpRect* | **LPRECT** | Points to a **RECT** data structure. |

### Return Value

None.

**See Also**

**GetClientRect** (in Windows SDK)

**VBGetControlRect**

## Example

```
VBGetClientRect(hctl, &rcControl);
```

## VBGetControl [2.0]

See Also Example

### Syntax

**HCTL VBGetControl**(*hctl*, *gc*)

Searches for a handle to a control based on a specified control. The *gc* value identifies the relationship between the two controls.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *gc* | **WORD** | Specifies the relationship between the original control and the returned control. It may be one of the values listed in the following table. |

| Value | Meaning |
|---|---|
| GC_FIRSTSIBLING | Returns the first sibling control. |
| GC_LASTSIBLING | Returns the last sibling control. |
| GC_NEXTSIBLING | Returns the next sibling control. |
| GC_PREVSIBLING | Returns the previous sibling control. |
| GC_CHILD | Returns the first child control. |
| GC_CONTAINER | Returns controls container, or NULL if none. |
| GC_FORM | Returns controls form. |
| GC_FIRSTCONTROL | Returns first control in enumeration. |
| GC_NEXTCONTROL | Returns next control in enumeration. |
| GC_FIRSTSELECTED | Returns first selected control in enumeration. |
| GC_NEXTSELECTED | Returns next selected control in enumeration. |

### Comments

When you enumerate controls, be aware of changes to the control. For example, changing the z-order of a control during enumeration could cause infinite looping.

### Return Value

The return value identifies a control. It is NULL if it reaches the end of a controls list of controls, or if the *gc* parameter is invalid.

**See Also**

**GetWindow** (in Windows SDK)

**Example**

```
// Enumerate controls in hctlCurrent's container.
hctl = VBGetControl(hctlCurrent, GC_FIRSTSIBLING);
do {
  // Process control
  ...
  hctl = VBGetControl(hctl, GC_NEXTSIBLING);
} while (hctl);
```

## VBGetControlHwnd

See Also Example

### Syntax

**HWND VBGetControlHwnd**(*hctl*)

Gets the controls window handle (HWND). Since a control procedure is passed both an HCTL and an HWND value, this function is usually not needed in custom control code. However, it can be useful when dealing with other controls, and in DLL routines that are passed an HCTL value.

When a Visual Basic statement passes a control as an argument, a DLL routine receives the handle to the control structure. The DLL routine may pass this handle to **VBGetControlHwnd** to get the controls window handle.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control structure. |

### Comments

Graphical controls do not have an HWND.

### Return Value

The window handle of the control, or NULL if a graphical control.

**See Also**

**VBGetControlModel**

**VBGetHwndControl**

## Example

```
hwnd = VBGetControlHwnd(hctl);
```

## VBGetControlModel

See Also Example

### Syntax

**LPMODEL VBGetControlModel**(*hctl*)

Gets a pointer to a control model. The control model is the same structure passed to **VBRegisterControl**. By examining fields of this structure you can determine how a controls flags are set, and you also have access to its property and event information tables.

This function may be useful in DLL routines which are passed a control. (When a Visual Basic statement passes a control as an argument, the DLL routine receives the handle to the control structure.)

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control structure. |

### Return Value

A far pointer to the control model.

**See Also**

**VBDerefControl**
**VBGetControlHwnd**

**Example**

The following function takes a handle for any valid control and a standard property value, and returns the index of that property if it is supported. Otherwise, it returns 1.

```
USHORT   GetStdPropIndex(HCTL hctl, PPROPINFO StdProp)
{
  MODEL     FAR *lpmodel;
  PPROPINFO   FAR *pPropinfo;
  LPSTR     lpstrTmp;

  lpmodel = VBGetControlModel (hctl);
  pPropinfo = (LPVOID)MAKELONG(lpmodel->npproplist,
      (_segment)lpmodel);

  for (; *pPropinfo; pPropinfo++) {
    if (StdProp == *pPropinfo)
      return pPropinfo - lpmodel->npproplist;
  }

  return -1;
}
```

## VBGetControlProperty

See Also Example

### Syntax

**ERR VBGetControlProperty**(*hctl*, *iProp*, *lpdata*)

Retrieves the current setting of a property. The function either reads data from the control structure or sends a VBM_GETPROPERTY message, depending on how the property is declared. Visual Basic calls this function before displaying a setting in the Properties window, and before returning a property value in Visual Basic code.

This function should not be used to process a VBM_GETPROPERTY message for the indicated property, because that can only result in an infinite loop. Instead, this function is useful in the following situations: when getting the value of a property from another control; when you need to know the value of a standard property; and within a DLL routine.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control structure. |
| *iProp* | **USHORT** | Index of the property within the controls property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lpdata* | **LPVOID** | Pointer to a location to place the data. If the PF_fPropArray flag is set, this parameter points to a DATASTRUCT structure. |

### Comments

If the property is a string property, *lpdata* must point to an HSZ variable or field. The function **VBGetControlProperty** creates an HSZ string and assigns the handle to the location pointed to by *lpdata*. This string is intended to be temporary, so it is the callers responsibility to free memory by calling **VBDestroyHsz** after using the string.

In the case of X, Y coordinate types, the function converts from twips to the scale of the controls container, as the value is retrieved. In the case of DT_BOOL types, nonzero values are converted to 1.

If the property is a picture property (type DT_PICTURE), the function creates an HPIC handle and assigns it to the *lpdata* location. The picture has a reference count of zero and does not need to be explicitly freed.

If the property is an enumerated property (type DT_ENUM), then *lpData* points to a BYTE.

### Return Value

Zero, if no error, or an error code.

**See Also**

VBCreateHsz
VBDestroyHsz
VBM_GETPROPERTY
VBSetControlProperty

**Example**

```
VBGetControlProperty(hctl, IPROP_CNTR_BACKCOLOR, &colorTemp);
VBGetControlProperty(hctl, IPROP_PUSH_CAPTION, &hsz);
```

## VBGetControlRect [2.0]

See Also

### Syntax

**VOID VBGetControlRect**(*hctl*, *lpRect*)

Copies the dimensions of the bounding rectangle of the specified control into the structure pointed to by the *lpRect* parameter. The dimensions are given in screen coordinates, relative to the upper-left corner of the display screen.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *lpRect* | **LPRECT** | Points to a **RECT** data structure that contains the screen coordinates of the upper-left and lower-right corners of the control. |

### Comments

This function is similar to the Windows API **GetWindowRect** function, except that the **VBGetControlRect** function can be used by graphical controls, whereas **GetWindowRect** cannot.

### Return Value

None.

**See Also**

**GetWindowRect** (in Windows SDK)
**VBGetClientRect**
**VBGetRectInContainer**

## VBGetDataSourceControl [3.0]

### Syntax

**HCTL VBGetDataSourceControl**(*hctl*, *bIsRegistered*)

Retrieves the hctl of the data control that the control is currently bound to.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *bIsRegistered* | **BOOL** | Specifies whether the bound control has completed all its binding initialization with the data control. |

### Comments

In most cases, a bound control responds to data binding messages in which one of the arguments is a pointer to a DATAACCESS structure. This structure contains the hctl of the data control. In cases where this structure is not available, you need to call the **VBGetDataSourceControl** function to return the hctl of the data control.

**Note** The hctl of the data control can change while the program runs. This means that you should always call this function right before you reference the data control, rather than using a saved copy of the hctl.

### Return Value

Hctl of data control, or NULL if the bound control is not currently bound to a data control.

## VBGetHInstance

Example

### Syntax

**HANDLE VBGetHInstance**(**VOID** )

Gets the instance handle for the currently executing .EXE file. In the development environment, this file is VB.EXE. When a stand-alone executable is being run, this file is the application itself.

The instance handle is necessary for some Windows calls, including **CreateWindow**, **RegisterClass**, and **UnregisterClass**.

### Return Value

An instance handle for the executable file.

**Example**

```
class.hInstance = VBGetHInstance();
```

## VBGetHlstr [2.0]

See Also Example

### Syntax

**USHORT VBGetHlstr**(*hlstr*, *pb*, *cbLen*)

Copies the string data from the given source string to the destination buffer. The string is null-terminated in the buffer. If the length of the source string is greater than *cbLen*, the source string is truncated to fit in the destination buffer.

| Parameter | Type | Description |
|---|---|---|
| *hlstr* | **HLSTR** | Handle to the source Basic language string. |
| *pb* | **LPVOID** | Far pointer to the destination buffer containing the copied null-terminated string. |
| *cbLen* | **USHORT** | The maximum number of bytes to copy to *pb*. If set to zero, *pb* is not referenced. |

### Comments

Basic language strings can contain embedded null characters.

If *hlstr* is a temporary Basic string (created by **VBCreateTempHlstr**), the string is deleted after it is copied to the destination buffer. To simply delete a temporary string, call **VBGetHlstr** with *pb* set to NULL and *cbLen* set to 0. To delete a string that is not a temporary string, use **VBDestroyHlstr**.

### Return Value

An unsigned integer value that contains the actual number of characters copied into *pb*. The terminating null character is not included in *cbLen*.

**See Also**

**VBCreateHlstr**
**VBDerefHlstr**
**VBDestroyHlstr**
**VBGetHlstrLen**

## Example

```
cbCount = VBGetHlstr(hlstr, &cBuffer, sizeof (Buffer) - 1);
```

## VBGetHlstrLen

See Also Example

### Syntax

**USHORT VBGetHlstrLen**(*hlstr*)

Gets the length of a Visual Basic language string. (Use **VBGetHlstr** to copy the string.)

| Parameter | Type | Description |
|---|---|---|
| *hlstr* | **HLSTR** | Handle to the Basic language string. |

### Comments

This function does not free temporary strings.

### Return Value

The length of the string in bytes.

**See Also**

**VBCreateHlstr**
**VBDerefHlstr**
**VBFireEvent**
**VBGetHlstr**

**Example**

```
lpstr[VBGetHlstrLen(params.ClickString) - 1] = '\0';
SetWindowText(hwnd, lpstr);
```

## VBGetHwndControl

See Also Example

### Syntax

**HCTL VBGetHwndControl**(*hwnd*)

Gets the handle to a control structure (HCTL), given the window handle (HWND) of that control. The control structure contains a good deal of information that the window structure does not, including the value of properties. Though the window handle is necessary for many Windows API calls, the handle to the control structure is necessary for many calls to the Visual Basic API.

| Parameter | Type | Description |
|---|---|---|
| *hwnd* | **HWND** | Window handle of a Visual Basic control. |

### Return Value

The handle of the control, or NULL if the window handle is invalid or not associated with a Visual Basic control.

**See Also**

VBGetControlHwnd

## Example

```
hctl = VBGetHwndControl(hwnd);
```

## VBGetMode

See Also Example

### Syntax

**MODE VBGetMode**(**VOID**)

Indicates the current mode   run, design, or break. When the control is used within the development environment, Visual Basic can be in any of the three modes. When the control is used within a stand-alone executable, Visual Basic is always in run mode.

Calling this function is useful when you want to alter behavior depending on the mode. For example, a control such as the timer should be visible in design mode even though it is not visible in run mode. The code that responds to VBM_CREATED can call this function to decide whether or not to display the control.

### Return Value

One of the following enumerated constants: MODE_DESIGN, MODE_RUN, or MODE_BREAK.

**See Also**

VBM_CREATED

## Example

The following code causes a control to be shown in design mode, but not break mode or run mode (the Visual Basic 2.0 flag, MODEL_fInvisAtRun now provides this functionality). The code responds to the VBM_CREATED message by testing the mode and returning immediately if Visual Basic is not in design mode. (Note that MODEL_fLoadMsg must be set to receive this message.) This prevents the default message processor from showing the control, which it normally does in response to VBM_CREATED.

```
case VBM_CREATED:
  if (VBGetMode() != MODE_DESIGN)
    return 0L;
```

## VBGetPIC

See Also          Example

### Syntax

**HPIC VBGetPic**(*hpic*, *lppic*)

Gets information from an internal picture structure and fills a PIC structure with this information. Calling this function allows you to use data referenced by a valid HPIC handle.

**Note** Given a choice, the **VBGetPicEx** function is preferred over the **VBGetPic** function.

| Parameter | Type | Description |
|---|---|---|
| *hpic* | **HPIC** | Handle to an internal picture structure. |
| *lppic* | **LPPIC** | Far pointer to a PIC structure to receive the information. |

### Comments

Conversion of data between PIC and HPIC data types is necessary. Although Visual Basic keeps track of picture data with a handle (HPIC), you have direct access only to the PIC type. Thus, when setting a DT_PICTURE property, Visual Basic supplies an HPIC; you need to call **VBGetPic** to dereference the data.

This function does not initialize the *hpal* member of the PIC structure. Use the **VBGetPicEx** function for this.

### Return Value

The HPIC that was passed to the function.

**See Also**

**VBAllocPicVBRefPic**

**VBFreePic**

**VBPicFromCF**

**Example**

```
VBGetPic(hpic, &pic);
switch (pic.picType) {
  case PICTYPE_BITMAP:
    GetObject(pic.picData.bmp.hbitmap, sizeof(BITMAP),
        (LPSTR)&bmp);
    hdcMem = CreateCompatibleDC(hdc);
    SelectObject(hdcMem, pic.picData.bmp.hbitmap);
    GetClientRect(hwnd, &rect);
    StretchBlt(hdc, 0, 0, rect.right, rect.bottom, hdcMem, 0,
    ú 0, bmp.bmWidth, bmp.bmHeight, SRCCOPY);
    DeleteDC(hdcMem);
    break;
...
}
```

## VBGetPicEx [2.0]

See Also Example

### Syntax

**HPIC VBGetPicEx**(*hPic*, *lpPic*, *usVersion*)

This function is almost identical to the **VBGetPic** function. The only difference is that the PIC data structure in Visual Basic version 2.0 contains an *hpal* member, and this function correctly fills in the *hpal* member. Refer to **VBGetPic** for more information.

| Parameter | Type | Description |
|---|---|---|
| *hPic* | **HPIC** | Handle to an internal picture structure. |
| *lpPic* | **LPPIC** | Far pointer to PIC structure. |
| *usVersion* | **USHORT** | The Visual Basic version. |

### Return Value

The allocated HPIC handle, or NULL if an error occurs.

**See Also**

VBAllocPicEx
VBFreePic
VBPicFromCF
VBRefPic

## Example

```
VBGetPicEx(hPic, &pic, VB_VERSION);

switch (pic.picType) {
  case PICTYPE_BITMAP:
    // Get HBMP and HPAL from PIC.
    hBmp = pic.picData.bmp.hbitmap;
    hPal = pic.picData.bmp.hpal;
... }
```

## VBGetRectInContainer [2.0]

See Also    Example

### Syntax

**VOID VBGetRectInContainer**(*hctl*, *lpRect*)

Copies the dimensions of the bounding rectangle of the specified control into the structure pointed to by the *lpRect* parameter. The dimensions are given in the coordinates of the controls container, relative to the upper-left corner of the display screen.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *lpPt* | **LPRECT** | Points to a **RECT** data structure that contains the screen coordinates of the upper-left and lower-right corners of the controls container. |

### Return Value

None.

**See Also**

**VBGetControlRect**

## Example

```
 VBGetRectInContainer(hctl, &rcControl);
```

## VBGetVariantType [2.0]

See Also Example

### Syntax

**SHORT VBGetVariantType**(*lpVar*)

Returns the Variant data type for the given Variant.

| Parameter | Type | Description |
|---|---|---|
| *lpVar* | **LPVAR** | Far pointer to Variant. |

### Comments

The function can fail if the Variant is not valid, or if the Variant references an object that is unable to get its value.

### Return Value

If the function fails, it returns 1. Otherwise, the function returns the Variant data type for *lpVar*.

| Return value | Meaning |
|---|---|
| VT_EMPTY | Empty |
| VT_NULL | Null |
| VT_I2 | Integer |
| VT_I4 | Long |
| VT_R4 | Single |
| VT_R8 | Double |
| VT_CURRENCY | Currency |
| VT_DATE | Date |
| VT_STRING | String |

**See Also**

**VBCoerceVariant**
**VBGetVariantValue**
**VBSetVariantValue**

**Example**

```
switch (VBGetVariantType(lpVar)) {
  case VT_I2:
    {
    int i2;

    if (VBGetVariantValue(lpVar, &i2) != -1)
      {
      // Integer processing
      }
    break;
...}
```

## VBGetVariantValue [2.0]

See Also

### Syntax

**SHORT VBGetVariantValue**(*lpVar*, *lpVal*)

Gets the value of the given Variant.

| Parameter | Type | Description |
|---|---|---|
| *lpVar* | **LPVAR** | Far pointer to Variant. |
| *lpVal* | **LPVOID** | Far pointer to data buffer. Buffer must be large enough to store the Variants value. |

### Comments

If the data type of the Variant is known (by calling **VBGetVariantType** first), *lpVal* need only point to a buffer large enough for the data type. Otherwise, *lpVal* should point to a data structure that is a union, 8 bytes in size, of all the types that you want to handle.

The function can fail if the Variant is not valid, or if the Variant references an object that is unable to get its value.

### Return Value

The function returns 1 if it fails. Otherwise, the function returns the Variant data type for *lpVar*.

**See Also**

**VBCoerceVariant**
**VBGetVariantType**
**VBSetVariantValue**

## VBGetVersion [2.0]

See Also

### Syntax

**SHORT VBGetVersion**()

Returns the version level of the host environment.

---

**Note** The function returns the correct value (VB100_VERSION) for Visual Basic version 1.0 (VBRUN100.DLL), even though **VBGetVersion** is a new function for Visual Basic version 2.0.

---

### Return Value

The version level.

| Value | Meaning |
|---|---|
| VB100_VERSION | Visual Basic version 1.0 |
| VB200_VERSION | Visual Basic version 2.0 |

**See Also**

**VBGetControlModel**

## VBInvalidateRect [2.0]

See Also

### Syntax

**VOID VBInvalidateRect**(*hctl*, *lpRect*, *bErase*)

Adds a rectangle to the update region of the controls window. The update region represents the client area of the window that must be redrawn. For graphical controls, the rectangle is added to the update region of the controls container.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *lpRect* | **LPRECT** | Points to a **RECT** structure that contains the rectangle to be added to the update region in client coordinates. If the *lpRect* parameter is NULL, the entire client area is added to the update region. |
| *fErase* | **BOOL** | Specifies whether the background within the update region is to be erased when the update region is processed. It this parameter is TRUE, the background is erased when the **BeginPaint** function is called. If this parameter is FALSE, the background remains unchanged. |

### Comments

If the *fErase* parameter is TRUE for any part of the update region, the background is erased in the entire region, not just in the given part.

The value of *lpRect* is always in the controls client coordinate system.

### Return Value

None.

**See Also**

**InvalidateRect** (in Windows SDK)

**VBUpdateControl**

## VBIsControlEnabled [2.0]

See Also

### Syntax

**BOOL VBIsControlEnabled**(*hctl*)

Determines whether the given control is enabled for mouse and keyboard input.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |

### Return Value

The return value is nonzero if the control is enabled. Otherwise, it is zero.

**See Also**

**IsWindowEnabled** (in Windows SDK)

## VBIsControlVisible [2.0]

See Also

### Syntax

**BOOL VBIsControlVisible**(*hctl*)

Determines the visibility state of the given control.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |

### Return Value

The return value is nonzero if the specified control is visible on the screen. The return value is zero if the control or any of its containers are not visible. The return value may be nonzero even if the window is totally obscured by other windows.

**See Also**

**IsWindowVisible** (in Windows SDK)

## VBLinkMakeItemName [2.0]

### Syntax

**VOID VBLinkMakeItemName**(*hctl*, *lpszBuff*)

Returns the name of the control with any control array information appended to it.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *lpszBuff* | **LPSZ** | Buffer that receives control name and any control array information. |

### Comments

The control array index is appended to the name passed in *lpszBuf*. For example, if *lpszBuff* contains foo and *hctl* is an element of a control array whose index is 3, then **VBLinkMakeItemName** returns the string foo(3). If *hctl* is not an element of a control array, the function returns foo. The *lpszBuff* parameter should be at least MAXLINKITEMNAME in length.

### Return Value

None.

## VBLinkPostAdvise [2.0]

See Also

### Syntax

**ERR LinkPostAdvise**(*hctl*)

Sends notification to the DDE client that the linked data has been changed by the DDE server (custom control).

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |

### Comments

The control should call the **VBLinkPostAdvise** function each time the linked data has changed. This allows the DDE client to get the most recent copy of the linked data.

If there is no DDE link, the function does nothing.

### Return Value

ERR value.

**See Also**

**VBPasteLinkOk**

## VBLockHsz

See Also Example

### Syntax

**LPSTR VBLockHsz** (*hsz*)

Locks the position of an HSZ string maintained by Visual Basic, so that it cannot be moved in memory, and returns a far pointer to the null-terminated string data.

Any time you need to pass the address of a string to a Visual Basic API function, you should use **VBLockHsz** to dereference the string. The **VBDerefHsz** function also can be used to dereference the string handle, but it does not lock the strings address as does **VBLockHsz**. If you dont lock the string, it is liable to be moved in memory before the address is used in the Visual Basic API function, thus causing that function to get invalid data.

After you are done using the string data, make sure you unlock the string by calling **VBUnlockHsz**.

| Parameter | Type | Description |
|---|---|---|
| *hsz* | **HSZ** | Handle to the string. |

### Return Value

A far pointer to the null-terminated string.

**See Also**

**VBCreateHsz**
**VBDerefHsz**
**VBUnlockHsz**

**Example**

```
lpstr = VBLockHsz(hsz);
cbStr = lstrlen(lpstr);
hlstr = VBCreateHlstr(lpstr, cbStr);
VBUnlockHsz(hsz);
```

## VBMoveControl [2.0]

See Also

### Syntax

**VOID VBMoveControl**(*hctl*, *lpRect*, *fRepaint*)

Changes the position and dimensions of a control. The position and dimensions values are relative to the upper-left corner of the client area of the controls container.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *lpRect* | **LPRECT** | New bounding coordinates of control. |
| *fRepaint* | **BOOL** | Specifies whether the window is to be repainted. |

### Comments

If the *fRepaint* parameter is FALSE, no repainting of any kind occurs. When this parameter is FALSE, the application must explicitly invalidate or redraw any parts of the control.

This function is similar to the Windows API **MoveWindow** and **SetWindowPos** functions, except that the **VBMoveControl** function can be used by graphical controls, whereas the Windows functions cannot.

### Return Value

None.

**See Also**

**MoveWindow** (in Windows SDK)
**SetWindowPos** (in Windows SDK)

## VBPaletteChanged [2.0]

See Also

### Syntax

**VOID VBPaletteChanged**(*hctl*)

Notifies Visual Basic that its palette has changed, thereby updating the current system palette.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |

### Comments

The **VBPaletteChanged** function causes Visual Basic to asynchronously reconstruct the system palette for the top-level window containing the control, ultimately resulting in VBM_PALETTECHANGED messages sent to palette-aware controls enumerated in z-order. This function has no effect if the control is not descended from the active window.

### Return Value

None.

**See Also**

VBSet Control Flags

## VBPasteLinkOk [2.0]

See Also

### Syntax

**BOOL VBPasteLinkOk**(*phTriplet*, *hctl*)

This function is a design-time only function. This function should be called only when a custom control receives a VBM_QPASTEOK message with *wp* = PT_PASTELINK, and the control is interested in becoming a client in a DDE conversation.

| Parameter | Type | Description |
|---|---|---|
| *phTripletl* | **HANDLE FAR *** | Pointer to a global memory handle or NULL. |
| *hctl* | **HCTL** | Handle to the control. |

### Comments

The *phTriplet* parameter is optional, and the control can pass NULL if it does not want a string returned.

If *phTriplet* is a pointer to a handle, the DDE server allocates a null-terminated string that contains the DDE triplet of application, topic, and item name. For example, excel!sheet1!r1c1 is a possible DDE triplet for Microsoft Excel. The control is responsible for freeing the returned handle.

### Return Value

TRUE indicates link request was accepted; FALSE indicates link was terminated.

**See Also**

**VBLinkPostAdvise**

## VBPicFromCF

See Also Example

### Syntax

**ERR VBPicFromCF**(*lphpic*, *hData*, *wFormat*)

Allocates an internal picture structure from the given Clipboard data handle and Clipboard format, and supplies a handle to the picture that was allocated.

The following Clipboard formats are supported: bitmap, metafile, palette, and device-independent bitmap (DIB).

| Parameter | Type | Description |
|---|---|---|
| *lphpic* | **HPIC FAR *** | Pointer to an HPIC data type; where the handle of the picture allocated should be copied. |
| *hData* | **HANDLE** | Handle to the Clipboard data. |
| *wFormat* | **WORD** | Integer representing Clipboard format. Accepted values include CF_BITMAP, CF_METAFILEPICT, CF_DIB, and CF_PALETTE. Icons are not supported with this operation. |

### Comments

If the *wFormat* parameter is CF_PALETTE, the **VBPicFromCF** function additionally creates a 1x1 pixel bitmap and stores the handle in the PIC structure.

### Return Value

Zero, if successful, or an error code.

**See Also**

**VBAllocPic**
**VBFreePic**
**VBGetPic**
**VBRefPic**

**Example**

```
if (OpenClipboard(hwnd)) {
  hData = GetClipboardData(CF_BITMAP);
  err = VBPicFromCF(&hpic, hData, CF_BITMAP);
  CloseClipboard();
}
```

## VBReadBasicFile

See Also Example

### Syntax

**ERR VBReadBasicFile**(*usFileNo*, *pb*, *cb*)

Reads data from a file previously opened by the application developer. Regardless of how the file was opened, **VBReadBasicFile** reads data as raw bytes, performing no translations.

With the *usFileNo* parameter, you must supply the same number used by the application developer in an **Open #** statement. To get this number, you should establish a property, an argument to a function call, or another mechanism, so that the user can communicate the file number to be used.

| Parameter | Type | Description |
|---|---|---|
| *usFileNo* | **USHORT** | The integer specified by the application developer in a file **Open #** statement. |
| *pb* | **LPVOID** | Pointer to location which is to receive data read from the file. |
| *cb* | **WORD** | Number of bytes to transfer. |

### Return Value

Zero, if successful, or an error code.

**See Also**

**VBRelSeekBasicFile**
**VBSeekBasicFile**
**VBWriteBasicFile**

**Example**

```
char pStrBuf[256];

err = VBReadBasicFile(fileNo, (LPVOID)pStrBuf, 255);
```

## VBReadFormFile

See Also

### Syntax

**ERR VBReadFormFile(***hformfile*, *pb*, *cb***)**

Reads data from a form file. Use this function to load a single property from a form file during a form load. This function is needed only when the PF_fSaveMsg flag is set.

When a form is loaded from disk, every property of every control is initialized. For each property, Visual Basic either reads data from the disk directly (if the PF_fSaveData flag is set) or sends a VBM_LOADPROPERTY message (if the PF_SaveMsg flag is set). In the latter case, it is the controls responsibility to read data from the file by using **VBReadFormFile**.

The VBM_LOADPROPERTY message provides an HFORMFILE value. This is a handle to a structure that contains information on the file being loaded, including current position within the file. Passing this handle to **VBReadFormFile** enables Visual Basic to correctly read properties from a file previously saved with **VBWriteFormFile.**

| Parameter | Type | Description |
|---|---|---|
| *hformfile* | **HFORMFILE** | Handle to the form file. |
| *pb* | **LPVOID** | Pointer to the buffer into which to copy the data read. |
| *cb* | **WORD** | Number of bytes to read. |

### Comments

Whether you need to use this function depends entirely on whether you set the PF_fSaveMsg flag in the property information table. Most of the time, you shouldnt set this flag, because Visual Basic knows how to read and write all the standard data types to disk. Setting this flag is useful in any of the following situations: the property contains information not limited to a simple type; you want to save a group of related properties together; or the data is more efficiently stored in some intermediate format that the user does not see.

You generally dont need to set the PF_fSaveMsg flag to support saving and loading of a picture property, because Visual Basic automates this for you if the property is declared as type DT_PICTURE.

### Return Value

Zero, if successful, or an error code.

**See Also**

**VBRelSeekFormFile**
**VBSeekFormFile**
**VBWriteFormFile**
VBM_LOADPROPERTY

## VBRecreateControlHwnd

Example

### Syntax

**ERR VBRecreateControlHwnd**(*hctl*)

Destroys the window associated with the control and recreates it. In the process, this function saves and restores both font information and standard properties stored only in the window and which would otherwise be erased when the window is destroyed (this includes the Enabled, TabIndex, TabStop, and Visible properties).

---

**Note** The **VBRecreateControlHwnd** function should not be use for container controls, that is, controls that set the MODEL_fChildrenOk flag.

---

This function is useful for changing pre-HWND properties after the window structure has already been created, because some window styles cannot be altered without destroying the window. The **VBRecreateControlHwnd** function is provided to make this operation easier. Alternatives to calling this function include not reflecting changes to these properties at design time and specifying that the properties are read-only at run time.

As always, window styles affected by properties should be adjusted in response to the WM_NCCREATE message.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to a control structure. |

### Comments

Any custom properties that are stored only as part of the window state (and not explicitly stored in the programmer-defined structure) need to be saved and restored when you call the function **VBRecreateControlHwnd**. The **VBGetControlProperty** and **VBSetControlProperty** functions are useful for this purpose.

Use of the **VBRecreateControlHwnd** function can affect responses to other messages. When the window is being destroyed and recreated in response to this function, you may want to suppress actions taken in response to WM_NCDESTROY and WM_NCCREATE. (WM_NCDESTROY code typically frees handles; WM_NCCREATE code typically sets properties to default values.) You can react appropriately by setting a flag variable in your code and then checking this variable when responding to these messages.

### Return Value

Zero, if successful, or an error code.

**Example**

The following example recreates the window for a modified list control. The code saves the ListIndex property (which is part of the state of the window and is not stored elsewhere) and then restores it after the window is recreated.

```
VBGetControlProperty(hctl, IPROP_MYLIST_LISTINDEX, &iListIndex);
fRecreating = TRUE;
VBRecreateControlHwnd(hctl);
fRecreating = FALSE;
VBSetControlProperty(hctl, IPROP_MYLIST_LISTINDEX,   (LONG)iListIndex);
```

## VBRefPic

See Also Example

### Syntax

**HPIC VBRefPic**(*hpic*)

Increments the reference count for an HPIC handle, indicating that another property is using the image referred to by HPIC. Properly updating the reference count (by calling **VBRefPic** and **VBFreePic** as needed) is important, because the picture structure is deleted from memory as soon as the reference count reaches zero.

| Parameter | Type | Description |
|---|---|---|
| *hpic* | **HPIC** | Handle to an internal picture structure. |

### Comments

You need to call **VBRefPic** to update the reference count when you set PF_fSetMsg flag without setting the PF_fSetData flag.

You do not need to call **VBRefPic** when you set the PF_fSetData flag of a DT_PICTURE property unless the picture data is used in more than one place.

### Return Value

The HPIC handle that was passed to the function.

**See Also**

**VBAllocPic**
**VBFreePic**
**VBGetPic**
**VBPicFromCF**

**Example**

```
 VBGetPic (VBRefPic(hpic), &pic);
```

## VBRegisterModel

Example

### Syntax

**BOOL VBRegisterModel(***hmodDLL*, *lpmodel***)**

Registers a new control type. This action enables Visual Basic to support instances of the control, and (if the development environment is running) it expands the Toolbox, displaying an icon for the new control type.

This function should be called only in the Visual Basic entry point, VBINITCC, which Visual Basic calls while loading the custom control file.

| Parameter | Type | Description |
|---|---|---|
| *hmodDLL* | **HANDLE** | Handle to the DLL module (in the case of custom controls, the DLL module is the .VBX file). |
| *lpmodel* | **LPMODEL** | Pointer to the control MODEL structure. |

### Comments

The *hmodDLL* parameter can be obtained in the DLL entry point called by Windows. Once this value is obtained, it should be saved in a static variable, so that it can then be used in the **VBRegisterModel** call in VBINITCC.

### Return Value

TRUE, if registering the control was succesful; FALSE otherwise. (Failure to register a control may indicate that a control type with the same class name was already registered.)

### Example

The following source lines from CIRC3.C illustrate how a control can register two different models, depending on the version of the host.

```
BOOL FAR PASCAL _export VBINITCC(USHORT usVersion, BOOL fRuntime)
{
...
  // Register control(s).
  if (usVersion < VB_VERSION)
    return VBRegisterModel(hmodDLL, &modelCircle_Vb1);
  else
    return VBRegisterModel(hmodDLL, &modelCircle);
}
```

## VBReleaseCapture [2.0]

See Also

### Syntax

**VOID VBReleaseCapture**(**VOID** )

Releases the mouse capture and restores normal input processing to the control that captured the mouse. This function should be called after**VBSetCapture**, when the control no longer needs all mouse input.

### Return Value

None.

**See Also**

**VBGetCapture**
**VBSetCapture**
**ReleaseCapture** (in Windows SDK)

## VBRelSeekBasicFile [3.0]

See Also

### Syntax

**LONG VBRelSeekBasicFile**(*usFileNo*, *offset*)

Moves the current position within a data file forward or backward by the indicated distance. This function is the same as **VBSeekBasicFile**, except that the position given is relative to the current file position.

When using this function, make sure that you never seek past the end or before the beginning of a file.

| Parameter | Type | Description |
|---|---|---|
| *usFileNo* | **USHORT** | The integer specified by the application developer in a file **Open #** statement. |
| *offset* | **LONG** | Distance to move forward in the file, measured in bytes. Negative value moves file position backward. |

### Comments

Refer to the **VBReadBasicFile** and **VBWriteBasicFile** functions for more information on data files and how to get file numbers for them.

### Return Value

The new offset.

**See Also**

**VBReadBasicFile**
**VBSeekBasicFile**
**VBWriteBasicFile**

## VBRelSeekFormFile

See Also

### Syntax

**LONG VBRelSeekFormFile**(*hformfile*, *offset*)

Moves the current position within a form file forward or backward by the indicated distance. This function is the same as **VBSeekFormFile**, except that the position given is relative to the current file position.

When using this function, make sure that you never seek past the end or before the beginning of a file.

| Parameter | Type | Description |
|---|---|---|
| *hformfile* | **HFORMFILE** | Handle to form file. |
| *offset* | **LONG** | Distance to move forward in the file, measured in bytes. Negative value moves file position backward. |

### Comments

Refer to the **VBReadFormFile** and **VBWriteFormFile** functions for more information on form files and how to get handles to them.

### Return Value

The new offset.

**See Also**

VBReadFormFile
VBSeek FormFile
VBWriteFormFile

## VBResizeHlstr [2.0]

See Also

### Syntax

**ERR VBResizeHlstr**(*hlstr*, *newCbLen*)

Reallocates the size of the given string to the new size. The string must already exist.

| Parameter | Type | Description |
|---|---|---|
| *hlstr* | **HLSTR** | Handle to the source Basic language string. |
| *newCbLen* | **USHORT** | The new size for the string. The size must be between 0 and 65534. |

### Comments

If the string is lengthened, all existing string data is preserved, and the contents of the lengthened portions is undefined   it is not initialized to zero. You should also assume that the string has moved in memory, and use **VBDerefHlstr** to retrieve the string value.

### Return Value

The return value is 0 if successful, and ERR if not successful.

**See Also**

**VBCreateHlstr**
**VBDestroyHlstr**
**VBDerefHlstr**
**VBGetHlstrLen**
**VBSetHlstr**

## VBRestoreFPState [2.0]

See Also

### Syntax

**VOID VBRestoreFPState**(*lpBuff*)

Restores the saved floating-point state. Use the **VBCbSaveFPState** function to retrieve the current floating-point state.

| Parameter | Type | Description |
|---|---|---|
| *lpBuf* | **LPVOID** | A far pointer to a buffer that contains the current floating-point state. |

### Comments

Refer to the **VBCbSaveFPState** function for a more detailed description about floating-point states.

### Return Value

None.

**See Also**

**VBCbSaveFPState**

## VBRuntimeError [2.0]

See Also

### Syntax

**VOID VBRuntimeError**(*err*)

Generates a run-time error in Basic code. Execution does not return to the procedure that calls this function.

| Parameter | Type | Description |
|---|---|---|
| *err* | **ERR** | Basic run-time error code. |

**Warning** Do not use this function when processing messages in the control procedure. It is strictly for use by exported DLL functions that are called directly from Visual Basic code.

### Return Value

None.

**See Also**

**VBSetErrorMessage**

## VBScreenToClient [2.0]

See Also

### Syntax

**VOID VBScreenToClient**(*hctl*, *lpPt*)

Converts the given point from screen coordinates to client coordinates. The new coordinates are relative to the upper-left corner of the given controls client area.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *lpPt* | **LPPOINT** | Points to a **POINT** data structure that contains the coordinates to be converted. |

### Return Value

None.

**See Also**

VBClientToScreen

## VBSeekBasicFile [3.0]

See Also

### Syntax

**LONG VBSeekBasicFile**(*usFileNo*, *offset*)

Moves the current position within a data file. When using this function, make sure that you never seek past the end or before the beginning of a file.

| Parameter | Type | Description |
|---|---|---|
| *usFileNo* | **USHORT** | The integer specified by the application developer in a file **Open #** statement. |
| *offset* | **LONG** | Position to move to within the file, measured in bytes from beginning of the file. |

### Comments

Refer to the **VBReadBasicFile** and **VBWriteBasicFile** functions for more information on data files and how to get file numbers to them.

### Return Value

The new offset.

**See Also**

**VBReadBasicFile**
**VBRelSeekBasicFile**
**VBWriteBasicFile**

## VBSeekFormFile

See Also

### Syntax

**LONG VBSeekFormFile**(*hformfile*, *offset*)

Moves the current position within a form file. When using this function, make sure that you never seek past the end or before the beginning of a file.

| Parameter | Type | Description |
|---|---|---|
| *hformfile* | **HFORMFILE** | Handle to form file. |
| *offset* | **LONG** | Position to move to within the file, measured in bytes from beginning of the file. |

### Comments

Refer to the **VBReadFormFile** and **VBWriteFormFile** functions for more information on form files and how to get handles to them.

### Return Value

The new offset.

**See Also**

**VBReadFormFile**
**VBRelSeekFormFile**
**VBWriteFormFile**

## VBSendControlMsg

### Syntax

**LONG VBSendControlMsg**(*hctl*, *msg*, *wp*, *lp*)

Sends a message to another control. This function is similar to the Windows **SendMessage** function. However, when you want to send a message to a control, it is better to use **VBSendControlMsg** than **SendMessage**, because the former works even if the control is not yet fully loaded.

When you send a message directly to a control by using **SendMessage** and the controls window handle, Visual Basic intercepts the message and then passes it along to the control procedure. Calling **VBSendControlMsg** accomplishes the same result, but is slightly more efficient.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control structure. |
| *msg* | **USHORT** | Integer identifying the message to process. |
| *wp* | **USHORT** | Word parameter of the message. |
| *lp* | **LONG** | Long (32-bit) parameter of the message. |

### Return Value

A long integer. The meaning depends on the message that is being processed. See descriptions of return value for individual messages.

## VBSetCapture [2.0]

See Also Example

### Syntax

**VOID VBSetCapture**(*hctl*)

Sets the mouse capture to the control. *Mouse capture* is the ability to receive all mouse input and to block other objects from receiving mouse input. Only one control can have the mouse capture at any given time. If a control has the mouse captured, the control receives mouse input whether or not the cursor is within its borders.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control structure. |

### Comments

Use **VBReleaseCapture** when the control no longer needs to have the mouse captured.

### Return Value

None.

**See Also**

**VBGetCapture**

**VBReleaseCapture**

**SetCapture** (in Windows SDK)

**Example**

```
VBSetCapture(hctl);
.
. // Mouse processing
.
VBReleaseCapture();
```

## VBSetControlFlags [2.0]

See Also Example

### Syntax

**ULONG VBSetControlFlags**(*hctl*, *mask*, *value*)

Sets and returns the particular characteristics of a control.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *mask* | **ULONG** | Identifies which bits to modify. For a description of the *mask* values, see the following table. |
| *value* | **ULONG** | Identifies the new value for all the CTLFLG_ values, regardless of the *mask* value. |

| Value | Meaning |
|---|---|
| CTLFLG_BOUNDDATASET [3.0] | Used as a semaphore when setting CTLFLG_DATACHANGED. |
| CTLFLG_DATACHANGED [3.0] | Determines if the data in the bound control has changed since the last time the data was retrieved from the data control. |
| CTLFLG_GRAPHICALOPAQUE | Determines if the graphical control paints every pixel of its rectangular extent. |
| CTLFLG_GRAPHICALTRANSLUCENT | Determines if the graphical control manipulates pixels in a bit-wise manner. |
| CTLFLG_HASPALETTE | Determines if the control owns a palette. |
| CTLFLG_USESPALETTE | Determines if the control is palette-aware. |

### Comments

Here is additional information about each of the flags:

**CTLFLAG_BOUNDDATASET [3.0]**

Used as a semaphore to indicate that any setting of the DataChanged property or CTLFLG_DATACHANGED flag to TRUE is ignored until the semaphore is cleared. This may be necessary to avoid side-effects in which new data retrieved from the data control is set in the bound control, thereby causing the DataChanged property to be set to TRUE. Typically, only data explicitly changed by the user should cause the DataChanged property to be set to TRUE.

The CIRC3 sample uses the CTLFLG_BOUNDDATASET flag to prevent the DataChanged property from being changed when new data is set in the control:

```
VBSetControlFlags(hctl, CTLFLG_DATACHANGED, 0L);
VBSetControlFlags(hctl, CTLFLG_BOUNDDATASET, CTLFLG_BOUNDDATASET);
err = VBSetControlProperty(lpda->hctlBound, IPROP_CIRCLE_CAPTION,
                 lData);
VBSetControlFlags(hctl, CTLFLG_BOUNDDATASET, 0L);
```

If setting the Caption property triggered an event procedure and the users event procedure set the DataChanged property, setting the DataChanged property would be ignored until the semaphore (CTL_BOUNDDATASET) was cleared.

**CTLFLAG_DATACHANGED [3.0]**

Indicates that the data value in the bound control has changed since the last time the value was retrieved from the data control. This flag, along with the CTLFLG_BOUNDDATASET flag, is actually the internal representation of the DataChanged property.

If the user explicitly changes a bound property, the CTLFLG_DATACHANGED flag should be set. The CIRC3 sample illustrates this when the Caption property is changed by the user:

```
case VBM_SETPROPERTY:
  switch (wp) {
    case IPROP_CIRCLE_CAPTION:
      VBSetControlFlags(hctl, CTLFLG_DATACHANGED,
                          CTLFLG_DATACHANGED);
```

**CTLFLAG_GRAPHICALOPAQUE**

Applies only to graphical controls   this flag is ignored by nongraphical controls. This flag indicates that the control paints every pixel of its rectangular extent, meaning that this control does not behave as a transparent or translucent object.

Setting this flag, clears the CTLFLAG_GRAPHICALTRANSLUCENT flag.

**CTLFLAG_GRAPHICALTRANSLUCENT**

Applies only to graphical controls   this flag is ignored by nongraphical controls. This flag is set by default and indicates whether the control manipulates pixels in a bit-wise manner when it performs a drawing operation.

If this flag is cleared, then each pixel value within the bounding rectangle of the control must either be determined by the control or remain unchanged. Any drawing operation in which the control combines the pixels with a pattern requires that this flag remain set. Note that drawing icons requires that this flag be set, since icons can have inverse screen patterns.

Setting this flag clears the CTLFLAG_GRAPHICALOPAQUE flag.

Drawing operations performed during a VBM_PAINT message may require the CTLFLAG_GRAPHICALTRANSLUCENT flag to be set, depending on the drawing mode. (Refer to the Windows API **SetROP2** function for more information on setting drawing modes.)

| Drawing mode | CTLFLAG_GRAPHICALTRANSLUCENT |
|---|---|
| **R2_NOT** | TRUE |
| **R2_MERGEPENNOT** | TRUE |
| **R2_MASKPENNOT** | TRUE |
| **R2_MERGENOTPEN** | TRUE |
| **R2_MASKNOTPEN** | TRUE |
| **R2_MERGEPEN** | TRUE |
| **R2_NOTMERGEPEN** | TRUE |
| **R2_MASKPEN** | TRUE |
| **R2_NOTMASKPEN** | TRUE |
| **R2_XORPEN** | TRUE |
| **R2_NOTXORPEN** | TRUE |
| **R2_BLACK** | FALSE |
| **R2_WHITE** | FALSE |
| **R2_NOP** | FALSE |
| **R2_COPYPEN** | FALSE |
| **R2_NOTCOPYPEN** | FALSE |

Similarly, when you use the Windows API **BitBlt** function, you may need the CTLFLAG_GRAPHICALTRANSLUCENT flag to be set, depending on the raster op code.

| Raster op code | CTLFLAG_GRAPHICALTRANSLUCENT |
|---|---|
| DSTINVERT | TRUE |
| MERGEPAINT | TRUE |
| NOTSRCERASE | TRUE |

| | |
|---|---|
| PATINVERT | TRUE |
| PATPAINT | TRUE |
| SRCAND | TRUE |
| SRCERASE | TRUE |
| SRCINVERT | TRUE |
| SRCPAINT | TRUE |
| BLACKNESS | FALSE |
| MERGECOPY | FALSE |
| NOTSRCCOPY | FALSE |
| PATCOPY | FALSE |
| SRCCOPY | FALSE |
| WHITENESS | FALSE |

**CTLFLAG_HASPALETTE**

Indicates whether the control has a palette. Changing this flag from FALSE to TRUE, TRUE to FALSE, or TRUE to TRUE generates a call to **VBPaletteChanged**.

The **VBSetControlFlags** function ignores this flag if the display environment doesnt support a palette.

**CTLFLAG_USESPALETTE**

Indicates whether the control uses a palette. Refer to the **VBTranslateColor** function for more information on how color values are mapped to palette colors. Setting this flag does not generate an action.

The **VBSetControlFlags** function ignores this flag if the display environment doesnt support a palette.

**Return Value**

The current or new setting of the controls flags.

**See Also**

VBPaletteChanged

**Example**

Here are some examples of how to use the **VBSetControlFlags** function:

Set a control to be palette-aware:

VBSetControlFlags(hCtl, CTLFLG_USESPALETTE, CTLFLG_USESPALETTE);

1. Remove the palette-aware setting from a control:

```
VBSetControlFlags(hCtl, CTLFLG_USESPALETTE, 0L);
```

2. Retrieve the current setting only:

```
ulFlags = VBSetControlFlags(hCtl, 0L, 0L);
```

## VBSetControlProperty

See Also Example

### Syntax

**ERR VBSetControlProperty**(*hctl*, *iProp*, *Data*)

Sets the value of a property. Depending on the setting of the propertys flags, this function may do any of the following: send a VBM_CHECKPROPERTY message; transfer data to the control structure; or send a VBM_SETPROPERTY message. Visual Basic calls this function after a setting is entered in the Properties window and after a setting is assigned in code.

This function should not be used to process these messages for the indicated property, because that results in an infinite loop. Use this function to: set the value of a property from another control; to set the value of a standard property; and to set the value of a property from within a DLL routine.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control structure. |
| *iProp* | **USHORT** | Index of the property within the controls property information table. The first property is indexed as 0, the second as 1, and so on. |
| *Data* | **LONG** | Data to store in the property. Although the data may have a type other than long integer, it must be placed in a 4-byte field and then passed as a long integer. If the PF_fPropArray flag is set, this parameter points to a DATASTRUCT structure. |

### Comments

To ensure that data is not altered when recast as a type Long, take the address of the data, cast that address to a pointer to a type Long, and then use indirection to obtain the original data. The first example below demonstrates this technique. Doing this is especially important with a floating-point number, because the format of the data completely changes if you use an ordinary (LONG) cast.

In the case of DT_BOOL types and X, Y coordinate types, the function converts the data if necessary. For X, Y coordinate types, the function assumes that units are expressed in the scale of the controls container. Then the scales are converted to twips. For DT_BOOL types, nonzero values are converted to 1.

In the case of string data, the *Data* argument takes a far pointer to a null-terminated string.

If the property has type DT_ENUM (enumerated), cast the data to type Long. Note that the high word is entirely ignored, but that the high byte of the lower word must be set to zero or Visual Basic reports an error. This check is done because Visual Basic does not support a 1-byte type; therefore, enumerated types are treated as integers but tested to see if they are less than 256.

### Return Value

Zero, if no error, or an error code.

**See Also**

**VBGetControlProperty**

VBM_CHECKPROPERTY

VBM_SETPROPERTY

## Example

```
VBSetControlProperty(hctrl, IPROP_MYCTL_ANGLE, *(LPLONG)&MyRealNumber);
VBSetControlProperty(hctl, IPROP_MYCTL_CAPTION,
   (LONG)(LPSTR)"New Caption.");
```

## VBSetErrorMessage

Example

### Syntax

**ERR VBSetErrorMessage**(*errnum*, *lpszString*)

Sets the text of an error message, just before you return the error code. The error can either be an error defined by Visual Basic or a programmer-defined errror message. The latter should be in the range 20,000 to 29,999. Visual Basic specifically reserves this range for use by custom control writers and Visual Basic programmers. Use of other numbers is liable to conflict with current or future software.

When you set the text of a Visual Basic error message, the text you specify replaces the placeholder in the error message string, if any. When you set the text of a programmer-defined error message, the text you specify supplies the entire error message string.

For a list of trappable error messages defined by Visual Basic, see Visual Basic online Help. For more information on how the application developer can trap errors, see Chapter 10, Handling Run-Time Errors, in the Visual Basic *Programmers Guide*.

| Parameter | Type | Description |
|---|---|---|
| *errnum* | **ERR** | Integer specifying a trappable error message. |
| *lpszString* | **LPSTR** | Pointer to the string to be displayed. |

### Comments

If youre not setting error-message text, you dont need to call **VBSetErrorMessage**. If you do need to set error-message text (either a placeholder or the whole string, as described above), make sure you call **VBSetErrorMessage** every time you return that error to Visual Basic.

### Return Value

The first argument, which is the error number set.

**Example**

```
#define   ERR_NEGVALUE 20000
...
   case VBM_SETPROPERTY:
      switch (wp) {
         case IPROP_MYCTL_MYPROP:
            if (lp < 0)
               return VBSetErrorMessage(ERR_NEGVALUE,
                  "Negative setting not allowed for this property.");
```

## VBSetHlstr

See Also Example

### Syntax

**ERR VBSetHlstr(***phlstr*, *pb*, *cbLen***)**

Assigns a new string value to an existing Basic language string (HLSTR). The new string data can be shorter or longer than the existing string. Because the language string is managed as part of the Visual Basic string space, it is automatically moved around in memory as needed.

The first argument is not an HLSTR handle, but a pointer to the handle.

HLSTR handles in Basic user-defined types are initialized to 0. If such a field (an HLSTR field set to zero) is passed to **VBSetHlstr**, the function automatically assigns a new HLSTR handle to the field. Note that this works only with HLSTR handles that are part of a Basic user-defined type. In most cases, the handle itself will not be changed, only the string data.

| Parameter | Type | Description |
|---|---|---|
| *phlstr* | **HLSTR far *** | Pointer to an HLSTR (a memory location that stores a handle to a Basic language string). |
| *pb* | **LPVOID** | Pointer to the string data to assign to the Basic language string. If this pointer is NULL, then the string is uninitialized and the existing data is not preserved. |
| *cbLen* | **USHORT** | Length of the new string in bytes. If this argument is 0, the the *pb* argument is ignored. If this argument is 1, then *pb* is assumed to be an HLSTR. |

### Comments

This function can assign one string to another by setting *cbLen* to 1 and *pb* to the HLSTR of the source string. If the source HLSTR is a temporary string, it is freed afterwards.

### Return Value

Zero, if successful, or an error code.

**See Also**

**VBCreateHlstr**

**VBDerefHlstr**

**VBGetHlstrLen**

### Example

The following examples illustrate different ways of using **VBSetHlstr** to set a Basic language string:

```
VBSetHlstr(&hlstr, NULL, 0);       // Set string to empty string.

VBSetHlstr(&hlstr, NULL, 10);// Reserve 10 characters.
wsprintf(VBDerefHlstr(hlstr), "%9d", outputVal);

// Copy string.
VBSetHlstr(&hlstr, "Test String", strlen("Test String"));
VBSetHlstr(&hlstr, hlstr2, -1);    // Copy hlstr2 to hlstr.
```

## VBSetVariantValue [2.0]

See Also

### Syntax

**SHORT VBSetVariantValue**(*lpVar*, *vtype*, *lpData*)

Sets a Variant to a given data value based on the given Variant data type.

| Parameter | Type | Description |
|---|---|---|
| *lpVar* | **LPVAR** | Far pointer to Variant. |
| *vtype* | **SHORT** | Variant data type. |
| *lpData* | **LPVOID** | Far pointer to buffer, containing data of type *vtype*. |

### Comments

Refer to the **VBGetVariantType** function for a list of Variant data types.

If *vtype* is VT_STRING, then the *lpData* is a pointer to an HLSTR, not the HLSTR itself.

If *lpData* points to an HLSTR that is a temporary string, the string is freed afterwards.

### Return Value

If *vtype* is invalid, the function returns 1. For any other errors, the function returns ERR. A return value of 0 indicates success.

**See Also**

**VBCoerceVariant**
**VBGetVariantType**
**VBGetVariantValue**

## VBSuperControlProc

See Also

### Syntax

**LONG VBSuperControlProc**(*hctl*, *msg*, *wp*, *lp*)

Invokes the Windows procedure (WndProc) of the controls superclass, if any. Otherwise, it invokes **DefWndProc**. The superclass of a control is specified in the control model.

This function takes a control handle, looks up the corresponding window handle, and passes the window handle as part of the message.

Although **VBDefControlProc** usually forwards messages to the superclass, the function **VBSuperControlProc** is a more direct means. **VBSuperControlproc** bypasses the Visual Basic default processing and the message is guaranteed to reach the superclass.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to a control. |
| *msg* | **USHORT** | Integer identifying the message to process. |
| *wp* | **USHORT** | Word parameter of the message. |
| *lp* | **LONG** | Long (32-bit) parameter of the message. |

### Return Value

A long integer. The meaning depends on the message which is being processed: see descriptions of return value for individual messages.

**See Also**

**VBDefControlProc**

## VBTranslateColor [2.0]

See Also

### Syntax

**COLOR VBTranslateColor**(*hctl*, *clr*)

Converts a Visual Basic-encoded color to an RGB color value for the control.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. If NULL, color translation uses **GetSysColor** only. |
| *clr* | **COLOR** | RGB color value. |

### Comments

If the high-order bit of the *clr* parameter is set, the low-order word of *clr* is converted into an RGB color value using the Windows API **GetSysColor** function. If *hctl* is NULL, this is the only color translation performed.

If the control is palette-aware, then Visual Basic converts the RGB color value to a palette-relative RGB value. A control is palette-aware if it uses a palette (CTLFLG_USESPALETTE set) and has a palette (CTLFLG_HASPALETTE set). A graphical control is palette-aware if it uses a palette and either the control or its container has a palette.

### Return Value

The return value is an RGB color value.

**See Also**
 **VBSetControlFlags**

## VBUnlockHsz

See Also Example

### Syntax

**VOID VBUnlockHsz**(*hsz*)

Unlocks the position of an HSZ string. This enables Visual Basic to move the string around in memory, so that any pointer to the string data becomes invalid. The handle to the string, however, remains valid even though the data moves.

Though this function causes the address of the data to be invalid, it should be called as soon as you are done referring to the data, so that Visual Basic can efficiently manage memory. When you need to refer to the string data again, use **VBLockHsz** or **VBDerefHsz** to dereference the handle and get the address.

| Parameter | Type | Description |
|---|---|---|
| *hsz* | **HSZ** | Handle to the string. |

**See Also**

VBLockHsz

**Example**

```
lpstr = VBLockHsz;
cbStr = _strlen(lpstr);
hlstr = VBCreateHlstr(lpstr, cbStr);
VBUnlockHsz(hsz);
```

## VBUpdateControl [2.0]

See Also    Example

### Syntax

**VOID VBUpdateControl**(*hctl*)

Updates the client area of the given control by sending a WM_PAINT message to the control if the update region for the control is not empty. Graphical controls receive a VBM_PAINT message.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |

### Comments

For nongraphical controls, Windows sends a WM_PAINT message whenever the controls update region is not empty and there are no other messages in the application queue for that window.

For graphical controls, the container will be painted, followed by any graphical controls in the update region. Graphical controls receive a VBM_PAINT message.

### Return Value

None.

**See Also**

**UpdateWindow** (in Windows SDK)

**VBInvalidateRect**

**Example**

```
// Force repaint of entire control.
VBInvalidateRect(hCtl, NULL, NULL);
VBUpdateControl(hCtl);
```

## VBWriteBasicFile

See Also Example

### Syntax

**ERR VBWriteBasicFile**(*usFileNo*, *pb*, *cb*)

Writes data to a file previously opened by the application developer. Regardless of how the file was opened, **VBWriteBasicFile** writes out data as raw bytes, performing no translations.

With the *usFileNo* parameter, you must supply the same number used by the application developer in an **Open #** statement. To get this number, you should establish a property, an argument to a function call, or another mechanism, so that the user can communicate the file number to be used.

| Parameter | Type | Description |
|---|---|---|
| *usFileNo* | **USHORT** | The integer specified by the application developer in an **Open #** statement. |
| *pb* | **LPVOID** | Pointer to location of bytes to be written to the file. |
| *cb* | **WORD** | Number of bytes to transfer. |

### Return Value

Zero, if successful, or an error code.

**See Also**

**VBReadBasicFile**
**VBRelSeekBasicFile**
**VBSeekBasicFile**

**Example**

```
int cbSum;

err = VBWriteBasicFile(fileNo, (LPVOID)&cbSum, sizeof (cbSum));
```

## VBWriteFormFile

See Also

### Syntax

**ERR VBWriteFormFile(***hformfile*, *pb*, *cb***)**

Writes data to a form file. Each call to this function, which is needed only when the PF_fSaveMsg flag is set, should be used to store the value of one property as a form is being saved.

When a form is saved to disk, the value of every property of every control is copied to the file. For each property, Visual Basic either writes data to the disk directly (if the PF_fSaveData flag is set) or sends a VBM_SAVEPROPERTY message. In the latter case, it is the controls responsibility to write data to the file by using **VBWriteFormFile**.

The VBM_SAVEPROPERTY message provides an HFORMFILE value. This is a handle to a structure containing information on the file to which data is being written, including current position within the file. Passing this handle to **VBWriteFormFile** enables Visual Basic to correctly write to a file so that **VBReadFormFile** can read from it later.

| Parameter | Type | Description |
|---|---|---|
| *hformfile* | **HFORMFILE** | Handle to the form file. |
| *pb* | **LPVOID** | Pointer to the buffer from which to read the data. This data is then written to the file. |
| *cb* | **WORD** | Number of bytes to write. |

### Comments

Whether you need to use this function depends entirely on whether you set the PF_fSaveMsg flag in the property information table. Most of the time, you shouldnt set this flag, because Visual Basic knows how to read and write all the standard data types to disk. Setting this flag is useful in any of the following situations: the property contains information not limited to a simple type; you want to save a group of related properties together; or the data is most efficiently stored in some intermediate format that the user does not see.

You generally dont need to set the PF_fSaveMsg flag to support saving and loading of a picture property, because Visual Basic automates this for you if the property is declared as type DT_PICTURE.

### Return Value

Zero, if successful, or an error code.

**See Also**

**VBReadFormFile**
**VBRelSeekFormFile**
**VBSeekFormFile**
VBM_SAVEPROPERTY

## VBXPixelsToTwips, VBYPixelsToTwips

See Also	Example

### Syntax

**LONG VBXPixelsToTwips**(*pixels*)
**LONG VBYPixelsToTwips**(*pixels*)

Converts a measurement in pixels into a measurement in logical twips, in the horizontal (X) or vertical (Y) direction. See **VBXTwipsToPixels** for a discussion of pixels and twips.

| Parameter | Type | Description |
|---|---|---|
| *pixels* | **SHORT** | A measurement in pixels. |

### Return Value

The measurement in logical twips.

**See Also**

**VBXTwipsToPixels**
**VBYTwipsToPixels**

**Example**

```
CLICKINPARAMS params;
float   VBx, VBy;

VBx = (float)VBXPixelsToTwips(x);
params.X = &VBx;
VBy = (float)VBYPixelsToTwips(y);
params.Y = &VBy;
VBFireEvent(hctl, IEVENT_CIRCLE_CLICKIN, &params);
```

## VBXTwipsToPixels, VBYTwipsToPixels

See Also

### Syntax

**SHORT VBXTwipsToPixels**(*twips*)
**SHORT VBYTwipsToPixels**(*twips*)

Converts a measurement in logical twips into a measurement in pixels, in the horizontal (X) or vertical (Y) direction.

The ratio between logical twips and pixels varies from one users configuration to the next, so if you need to convert between the two, you should call this function rather than assuming a fixed ratio.

A pixel is the smallest unit of resolution on a display device. Windows GDI functions use this measurement by default. A twip is one-twentieth of a printers point: there are 1,440 twips to an inch and 567 twips to a centimeter. A *logical twip* corresponds to a length of one twip when Visual Basic prints a form, so this measurement is independent of the monitor and display device. Visual Basic controls use this measurement by default.

| Parameter | Type | Description |
|---|---|---|
| *twips* | **LONG** | A measurement in logical twips. |

### Return Value

The measurement in pixels.

**See Also**

VBXPixelsToTwips

VBYPixelsToTwips

## VBZOrder [2.0]

See Also

### Syntax

**VOID VBZOrder**(*hctl*, *zorder*)

Alters the drawing order of the control by putting it at either the top or bottom of the drawing order.

| Parameter | Type | Description |
|---|---|---|
| *hctl* | **HCTL** | Handle to the control. |
| *zorder* | **WORD** | Alters the drawing order of the control, as described in the following table. |

| Value | Meaning |
|---|---|
| ZORDER_BACK | Sends the control to the bottom of the drawing order. |
| ZORDER_FRONT | Brings the control to the top of the drawing order. |

### Comments

Within a container, all graphical controls remain below nongraphical (or windowed) controls. ZORDER_FRONT applied to a graphical control, brings it to the front of other graphical controls in that container. Similarly, ZORDER_BACK applied to windowed controls, moves the control behind other windowed controls, but always above all other graphical controls.

### Return Value

None.

**See Also**

VBM_METHOD

## VBM_CANCELMODE

Indicates that the internal state of the control should be reset. This message is typically sent when Visual Basic is taking away the mouse capture or moving the focus away from the control.

If the control records information about internal state (for example, if the control procedure has a flag indicating whether the control has captured the mouse or not), that information should be reset in response to this message. Otherwise, no response is usually needed. Visual Basic itself handles the implementation of changing the mouse capture or the focus.

Information about internal state concerns aspects of window operation; this information is normally distinct from property values.

| Parameter | Description |
|---|---|
| *wp* | The window handle of the control that has captured the mouse. |
| *lp* | Unused. |

### Default Action

Sends mouse-up messages, followed by a WM_CANCELMODE message, to the window that has captured the mouse. These messages use button coordinates that are outside of the client area and should therefore avoid spurious firing of a Click event.

The purpose of sending the mouse-up messages is to force subclassed controls to reset their internal state. If you are writing a control from scratch, you can safely process this message yourself and then return.

## VBM_CHECKPROPERTY

See Also

Requests that a property value be checked for validity. Visual Basic sends this message immediately before assigning new data to a property if that property is declared with the PF_fSetCheck flag set. The response to this message determines whether Visual Basic proceeds with the property assignment.

| Parameter | Description |
|---|---|
| *wp* | Index of the property within the controls property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lp* | Data to be assigned to the property if the check is successful. See VBM_SETPROPERTY for details on the data format. |

### Return Value

Zero, if the property assignment should proceed, or an error code.

### Default Action

Tests the data for validity for any standard property that can receive this message. If *wp* corresponds to a custom property, do not call **VBDefControlProc**.

**See Also**

**VBSetControlProperty**

VBM_SETPROPERTY

## VBM_COPY

See Also

Indicates that the control has just been copied to the Clipboard. Visual Basic sends this message in design mode, when the user has selected the Copy command from the Edit menu.

If you dont care about copying and pasting data between the control and other applications, you can ignore this message.

When this message is received, Visual Basic has already handled all the standard processing needed to copy the control to the Clipboard. This is a special format, supported just for controls, and it includes the values of all properties. However, this format is recognized only within Visual Basic.

The purpose of this message is to give your code an opportunity to copy some of the controls data to the Clipboard in other formats, so the data can be recognized by other applications. For example, when you copy a command button, the control itself is copied to the Clipboard. But in addition, the control responds to VBM_COPY by copying the Caption to the Clipboard in ordinary text format. This text, in turn, can be pasted into other applications.

| Parameter | Description |
|---|---|
| *wp* | Unused |
| *lp* | Unused |

### Comments

The code that responds to this message must open the Clipboard, copy data, and close the Clipboard. It must not empty the Clipboard. For an example, see **VBPicFromCF** in Chapter 12, Functions.

### Return Value

Zero, if successful, or an error code.

### Default Action

Returns zero.

**See Also**

**VBPicFromCF**

VBM_PASTE

VBM_QPASTEOK

## VBM_CREATED

See Also Example

Indicates that a control has just been created and that all properties have been loaded from disk or copied. The control may have been created through any number of means: through design-time drawing on a form; through loading from disk (as part of a form load); or through dynamic run-time creation via the Visual Basic **Load** statement.

This message is sent only if the MODEL_fLoadMsg flag is set in the control model. Otherwise, Visual Basic displays the control directly rather than sending this message first.

| Parameter | Description |
|---|---|
| *wp* | Unused |
| *lp* | Unused |

### Default Action

Displays the control, which results in WM_PAINT and WM_SIZE messages being sent. If the MODEL_fInvisAtRun flag is set, the control is displayed in design mode but not in run mode.

**See Also**

**VBGetMode**

VBM_INITIALIZE

**Example**

```
case VBM_CREATED:
  pcircle (PCIRCLE)VBDerefControl(hctl);

  // Test for user-defined flag.
  if (pcircle->fl & MYMODEL_fAutoSize)
    AutosizeMyControl(hctl);
  break;
```

## VBM_DATA_AVAILABLE [3.0]

See Also

Sent from the data control to the bound control whenever there is new data available in the current recordset. For example, if the data control moves to the next record, the VBM_DATA_AVAILABLE message is sent to all bound controls that are linked to the data control.

The typical response to a VBM_DATA_AVAILABLE message is to send back a VBM_DATA_GET message to the data control, indicating the specific data value that you want to retrieve. The CIRC3 sample illustrates how to do this.

| Parameter | Description |
|---|---|
| *wp* | Unused. |
| *lp* | A far pointer to a DATAACCESS structure. |

### Comments

After you set your property using the field value from the data control, you must set the DataChanged property to False. You can do this by using the **VBSetControlFlags** functions CTLFLG_DATACHANGED flag. Refer to Appendix D, Creating a Bound Custom Control, for sample code that implements the VBM_DATA_AVAILABLE message.

The *lp->sAction* value indicates what type of action triggered the VBM_DATA_AVAILABLE message. For example, if the user deletes an item from the current recordset, the *lp->sAction* value is set to DATA_DELETE

**Note** The VBM_DATA_AVAILABLE message is sent after the action specified by *lp->sAction* has completed.

The following table lists all the actions that generate this message:

| Action | Description |
|---|---|
| **DATA_ADDNEW** | New record prepared for insertion at the end of the recordset. |
| **DATA_BOOKMARK** | Bookmark set. |
| **DATA_CLOSE** | Recordset closed. |
| **DATA_DATAFIELDCHANGED** | DataField property has changed. |
| **DATA_DELETE** | Record deleted from recordset. |
| **DATA_FINDFIRST** | First record located that satisfies the specified criteria that record is now the current record. |
| **DATA_FINDLAST** | Last record located that satisfies the specified criteria that record is now the current record. |
| **DATA_FINDNEXT** | Next record located that satisfies the specified criteria that record is now the current record. |
| **DATA_FINDPREV** | Previous record located that satisfies the specified criteria that record is now the current record. |
| **DATA_MOVEFIRST** | Moved to the first record in the recordset. |
| **DATA_MOVELAST** | Moved to the last record in the recordset. |
| **DATA_MOVENEXT** | Moved to the next record in the recordset. |
| **DATA_MOVEPREV** | Moved to the previous record in the recordset. |
| **DATA_READDATA** | Bound control updated directly. Equivalent to *Data1*.UpdateControls. |
| **DATA_REFRESH** | New recordset created by the data control. Bound control needs to update all data it has about the recordset. |
| **DATA_ROLLBACK** | Changes reversed during the current transaction and transaction ended. |
| **DATA_SAVEDATA** | Recordset updated directly. Unlike **DATA_UPDATE**, does not trigger Validate event. Equivalent to *Data1*.UpdateRecord. |
| **DATA_UPDATE** | Recordset updated. Triggers Validate event. Equivalent to |

*Data1*.*recordset*.Update.

**Return Value**

Zero, if successful, or an error code.

**Default Action**

Sends a VBM_DATA_GET to the bound control to retrieve the newly available field value. The field value then becomes the value of the default property. If the data type of the field value is incompatible with the data type of the default property, Visual Basic generates a run-time error.

In addition, the default processing changes the DataChanged property to False. If *sAction* is DATA_CLOSE, DATA_DELETE, or DATA_UNLOAD, the default processing just clears the DataChanged property and returns -- it does not attempt to retrieve data.

**See Also**

VBM_DATA_GET

## VBM_DATA_GET [3.0]

See Also

Sent from the bound control to the data control in response to the bound control receiving the VBM_DATA_AVAILABLE message. This message allows the bound control to retreive field values and field attributes from the data control. Send this message using the **VBSendControlMsg** function.

Typically, the bound control fills in the necessary structure members of the DATAACCESS structure: *sAction*, *usDataType*, *sDataFieldIndex*, *hszDataField*, and so forth. The data control, in return, fills in the *lData* structure member with the desired data value.

This message can also be sent anytime by the bound control to get information from the data control. If you need to get the hctl of the data control, use the **VBGetDataSourceControl** function.

| Parameter | Description |
|---|---|
| *wp* | Unused. |
| *lp* | A far pointer to a DATAACCESS structure. |

### Comments

The *lp->sAction* value indicates what type of value to retrieve. For example, if you want to retrieve the current value of a particular field, set *lp->sAction* to DATA_FIELDVALUE The following table lists all the values that can be retrieved:

| Value | Description |
|---|---|
| **DATA_BOF** | Determines whether current record position is before the first record in the recordset. Returns a DT_SHORT. Equivalent to *Data1.recordset*.BOF. |
| **DATA_BOOKMARK** | Get a bookmark for the recordset according to the *lData* setting:<br>**DATA_BOOKMARKCURRENT** - return bookmark for current record.<br>**DATA_BOOKMARKFIRST** - return bookmark for first record.<br>**DATA_BOOKMARKLAST** - return bookmark for last record.<br>**DATA_BOOKMARKNEXT** - return bookmark for next record relative to bookmark in *hlstrBookMark*.<br>**DATA_BOOKMARKPREV** - return bookmark for previous record relative to bookmark in *hlstrBookMark*.<br>Returns an HLSTR. Equivalent to *Data1.recordset*.Bookmark. |
| **DATA_BOOKMARKABLE** | Determines whether the recordset supports bookmarks. Returns a DT_SHORT. Equivalent to *Data1.recordset*.Bookmarkable. |
| **DATA_EOF** | Determines whether current record position is after the last record in the recordset. Returns a DT_SHORT. Equivalent to *Data1.recordset*.EOF. |
| **DATA_FIELDATTRIBUTES** | Get attribute for the field. The returned *lData* value can be 0 or any combination of the following values:<br>**DB_FIXEDFIELD** - value in the field is fixed-length.<br>**DB_AUTOINCRFIELD** - value for new records is automatically incremented by the database.<br>**DB_UPDATABLEFIELD** - value in the field can be changed.<br>Returns a DT_LONG. Equivalent to *Data1.recordset*.Fields(*fieldname*).Attributes. |
| **DATA_FIELDCHUNK** | Get a chunk of data as specified by the *ulChunkOffset* and *ulChunkNumBytes* members of the DATAACCESS structure. Returns an HLSTR. Equivalent to *Data1.recordset*.Fields(*fieldname*).GetChunk. |
| **DATA_FIELDNAME** | Get the field name. Returns an HSZ. Equivalent to *Data1.recordset*.Fields(*fieldname*).Name. |
| **DATA_FIELDPOSITION** | Returns the ordinal position of the of the field in the Fields collection. Returns a DT_SHORT. Equivalent to *Data1.recordset*.Fields(*fieldname*).OrdinalPosition. |
| **DATA_FIELDSCOUNT** | Get the number of fields in the current recordset. Returns a DT_SHORT. Equivalent to *Data1.recordset*.Fields.Count. |
| **DATA_FIELDSIZE** | Get the field size in bytes. Returns a DT_LONG. Equivalent to *Data1.recordset*.Fields(*fieldname*).Size. |

| Value | Description |
|---|---|
| **DATA_FIELDTYPE** | Get the field data type. The returned *lData* value is:<br>**VT_DATA_BOOL** - True/False<br>**VT_DATA_VAR_BTYE** - Byte<br>**VT_DATA_INTEGER** - Integer<br>**VT_DATA_LONG** - Long<br>**VT_DATA_CURRENCY** - Currency<br>**VT_DATA_SINGLE** - Single<br>**VT_DATA_DOUBLE** - Double<br>**VT_DATA_DATETIME** - Date/Time<br>**VT_DATA_TEXT** - Text<br>**VT_DATA_BINARY** - Long Binary<br>**VT_DATA_MEMO** - Memo<br>The **DATA_FIELDTYPE** value returns the actual data type of the field as defined by the database. When you retrieve a field value from the data control, you set *usDataType* to the data type of the property that will contain the field value.<br>Refer to the definition of the Type property in the Visual Basic *Language Reference,* or online help, for information on the mapping between VT_DATA data types and Visual Basic data types.<br>Returns a DT_SHORT. Equivalent to *Data1*.*recordset*.Fields(*fieldname*).Type. |
| **DATA_FIELDVALUE** | Get the field value. Returns a value that is coerced, if possible, into the data type specified by *usDataType*. Equivalent to *Data1*.*recordset*.Fields(*fieldname*).Value. |
| **DATA_LASTMODIFIED** | Get the bookmark of the last modified record. Returns an HLSTR. Equivalent to *Data1*.*recordset*.LastModified. |
| **DATA_RECORDCOUNT** | Determines the number of records in the recordset. Returns a DT_LONG. Equivalent to *Data1*.*recordset*.RecordCount. |
| **DATA_UPDATABLE** | Determines whether the recordset can be updated. Returns a DT_SHORT. Equivalent to *Data1*.*recordset*.Updatable. |

### Comments

When using the **DATA_FIELDVALUE** action, a *lp->fs* value of DA_fNull indicates whether the retrieved value is a null. For numeric values, this allows you to distinguish between a null value and a 0 value. If *usDataType* is DT_HSZ and the field value is null, an HSZ containing a null string is returned in *lData*.

When using the **DATA_BOOKMARK** action, the bookmark is returned as an HLSTR in the *lp->lData* value. If sending the VBM_DATA_GET message returns an error, test the *lp->fs* value to determine the type of error: DA_fBOF indicates the current record is before the first record in the record set; DA_fEOF indicates the current record is after the last record in the recordset.

**Important** Any HLSTR returned in *lp->lData* is a permanent HLSTR. This means that HLSTRs are owned by the bound control and ***must*** be deallocated when no longer needed.

### Return Value

Zero, if successful, or an error code.

**See Also**

**VBGetDataSourceControl**
**VBSendControlMsg**
VBM_DATA_AVAILABLE

## VBM_DATA_METHOD [3.0]

See Also

Sent from the bound control to the data control whenever the bound control needs the data control to perform a desired method, such as moving to the next record in the current recordset. The VBM_DATA_METHOD is typically used in response to user interaction. For example, you might create a custom command button that automatically forces the data control to move to the next record. Send this message using the **VBSendControlMsg** function.

This message should never be sent in response to a VBM_DATA_AVAILABLE or VBM_DATA_REQUEST message.

You will need to get the hctl of the data control by using the **VBGetDataSourceControl** function.

| Parameter | Description |
|---|---|
| *wp* | Unused. |
| *lp* | A far pointer to a DATAACCESS structure. |

### Comments

The *lp->sAction* value indicates what type of method you want the data control to perform. For example, to force the data control to move to the next record, set *lp->sAction* to MOVENEXT. The following table lists all the methods that you can perform on the data control:

| Method | Description |
|---|---|
| **DATA_ADDNEW** | Prepare a new record to add to the recordset. |
| **DATA_BOOKMARK** | Move to the bookmark as specified by *lp->hlstrBookmark*. |
| **DATA_DELETE** | Delete the current record in the recordset. After deleting a record, you should move to another record, so that the user is not positioned on an invalid record. |
| **DATA_MOVEFIRST** | Move to the first record. |
| **DATA_MOVELAST** | Move to the last record. |
| **DATA_MOVENEXT** | Move to the next record. |
| **DATA_MOVEPREV** | Move to the previous record. |

The first four data values of the DATAACCESS structure are required for sending the VBM_DATA_METHOD message: *usVersion*, *sAction*, *hctlData* and *hctlBound*.

### Return Value

Zero, if successful, or an error code.

**See Also**

**VBGetDataSourceControl**
**VBSendControlMsg**

## VBM_DATA_REQUEST

See Also

Sent from the data control to the bound control whenever it needs to update the recordset with a value from the bound control. For example, if you modify the value of the bound property in your control and the data control moves to the next record, the VBM_DATA_REQUEST message is sent.

The typical response to a VBM_DATA_REQUEST message is to determine if the property value has changed. If it has, send back a VBM_DATA_SET message to the data control, indicating the specific data value that you want to set. (The CIRC3 sample illustrates how to use these messages).

| Parameter | Description |
|---|---|
| *wp* | Unused. |
| *lp* | A far pointer to a DATAACCESS structure. |

### Comments

The *lp->sAction* value indicates what type of action triggered the VBM_DATA_REQUEST. For example, if the user refreshes the current recordset, the *lp->sAction* value is set to DATA_REFRESH.

**Note** The VBM_DATA_REQUEST message is sent before the action specified by *lp->sAction* has completed.

The following table lists all the actions that generate this message:

| Action | Description |
|---|---|
| **DATA_ADDNEW** | Prepares a new record for insertion at the end of the recordset. |
| **DATA_BOOKMARK** | Sets a bookmark. |
| **DATA_CLOSE** | Closes a recordset. |
| **DATA_FINDFIRST** | Locates first record that satisfies the specified criteria and makes that record the current record. |
| **DATA_FINDLAST** | Locates last record that satisfies the specified criteria and makes that record the current record. |
| **DATA_FINDNEXT** | Locates next record that satisfies the specified criteria and makes that record the current record. |
| **DATA_FINDPREV** | Locates previous record that satisfies the specified criteria and makes that record the current record. |
| **DATA_MOVEFIRST** | Move to the first record in the recordset. |
| **DATA_MOVELAST** | Move to the last record in the recordset. |
| **DATA_MOVENEXT** | Move to the next record in the recordset. |
| **DATA_MOVEPREV** | Move to the previous record in the recordset. |
| **DATA_REFRESH** | New recordset created. Bound control needs to update all data values. |
| **DATA_SAVEDATA** | Updates data control directly. Unlike **DATA_UPDATE**, does not trigger Validate event. |
| **DATA_UNLOAD** | Unload form. |
| **DATA_UPDATE** | Updates recordset. Triggers Validate event. |

The data control relies on the bound control to tell it when a value needs to be updated. This can happen in two different ways. First, the application can set the bound controls DataChanged property to **True** to force an update. Second, the bound control can set the CTLFLG_DATACHANGED flag via the **VBSetControlFlags** function in response to the user setting the bound property.

The CIRC3 sample sets the CTLFLG_DATACHANGED whenever the Caption property is changed:

```
case VBM_SETPROPERTY:
  switch (wp) {
    case IPROP_CIRCLE_CAPTION:
      VBSetControlFlags(hctl, CTLFLG_DATACHANGED,
                 CTLFLG_DATACHANGED);
                 break;
...
```

### Return Value

Zero, if successful, or an error code.

### Default Action

If the DataChanged property is True, sends a VBM_DATA_SET message to the data control, passing the current value of the default property as the field value to set in the data controls recordset. If the data type of the default property value is incompatible with the data type of the field in the recordset, Visual Basic generates a run-time error.

**See Also**

VBM_DATA_SET

## VBM_DATA_SET [3.0]

See Also

Sent from the bound control to the data control in response to the bound control requesting new data values via the VBM_DATA_REQUEST message. This message allows the bound control to alter the data values in the recordset. Send this message using the **VBSendControlMsg** function.

Typically, the bound control fills in the necessary structure members of the DATAACCESS structure: *sAction*, *usDataType*, *sDataFieldIndex*, *hszDataField*, and so forth. The *lData* structure member contains the with the desired data value to send to the data control.

| Parameter | Description |
|---|---|
| *wp* | Unused. |
| *lp* | A far pointer to a DATAACCESS structure. |

### Comments

The *lp->sAction* value indicates whether the *lData* value is a single data value or part of a sequence of chunks of data. The following table lists the values that *lp->sAction* can have:

| Value | Description |
|---|---|
| **DATA_FIELDVALUE** | Set the field value. The *usDataType* member of the DATAACCESS structure specifies the data type of *lData*. Equivalent to *Data1*.*recordset*.Fields(*fieldname*).Value. |
| **DATA_FIELDCHUNK** | Set the field value using a chunk of data. This requires creating an HLSTR that contains the data and passing it as the *lData* value. Equivalent to *Data1*.*recordset*.Fields(*fieldname*).AppendChunk. |

**Note** If a temporary HLSTR is passed as the *lData* value, the data control will deallocate the HLSTR after processing it.

### Return Value

Zero, if successful, or an error code.

**See Also**

**VBSendControlMsg**

VBM_DATA_REQUEST

## VBM_DRAGDROP

Indicates that the control is the target of a drop operation. (A control is sent this message when another control is dropped on top of it.) Usually, there is no need to respond to this message because the default action is sufficient.

| Parameter | Description |
|---|---|
| *wp* | Unused. |
| *lp* | A far pointer to a DRAGINFO structure. |

### Return Value

Zero, if successful, or an error code.

### Default Action

The controls DragDrop event is fired if PEVENTINFO_STD_DRAGDROP is listed in the event information table.

## VBM_DRAGOVER

Indicates that the control is the target of a drag-over operation. (A control is sent this message when another control is dragged over it.) Usually, there is no need to respond to this message because the default action is sufficient.

| Parameter | Description |
|---|---|
| *wp* | Unused. |
| *lp* | A far pointer to a DRAGINFO structure. |

### Return Value

Zero, if successful, or an error code.

### Default Action

The controls DragOver event is fired if PEVENTINFO_STD_DRAGOVER is listed in the event information table.

## VBM_FIREEVENT

See Also

Indicates that a specified event should be fired now.

In most cases, a control procedure does not need to wait to receive this message before firing an event. The event can be fired any time an action is recognized. In other cases, you should defer firing an event such as a response to WM_SETFOCUS or WM_KILLFOCUS   thats what this message is used for.

Instead of firing the event, the control procedure posts a VBM_FIREEVENT message to itself, giving other pending messages a chance to be processed. When the VBM_FIREEVENT message is received, it can proceed with actually firing the event by calling **VBFireEvent**.

Two standard events that use this mechanism are GotFocus and LostFocus. Deferring either of these events allows Windows to take care of focus issues before Visual Basic statements are executed.

| Parameter | Description |
|---|---|
| *wp* | Index of the event in the controls event information table. The first event is indexed as 0, the second as 1, and so on. |
| *lp* | Unused, although the code that posts this message can use this parameter to point to an argument list. |

### Comments

When a custom event is involved, this message is sent only by a control procedure to itself. The control procedure can therefore use *lp* as needed; *lp* can be used to point to an argument list.

### Default Action

Fires the event with no parameters (other than *Index* when needed).

**See Also**

<u>**VBFireEvent**</u>

## VBM_GETDEFSIZE [2.0]

Example

Gets default size for a control in pixels. This size is used when a control is created by double-clicking on a control in the Toolbox window.

| Parameter | Description |
|---|---|
| *wp* | Unused. |
| *lp* | A far pointer to the controls MODEL structure. |

### Comment

This message is a model message; therefore, the control handle is NULL when the control procedure receives this message.

### Return Value

The exact width and height of the control in pixels. The low word of the return value is the width; the high word of the return value is the height.

### Default Action

Returns a fixed default size for all controls.

## Example

```
case VBM_GETDEFSIZE:
     return (MAKELONG(DEF_WIDGET_WIDTH, DEF_WIDGET_HEIGHT));
```

## VBM_GETPALETTE [2.0]

See Also

Requests a logical palette. This message is only sent to palette-aware controls.

| Parameter | Description |
|---|---|
| *wp* | Unused |
| *lp* | Unused |

### Return Value

The control should return an HPALETTE representing its desired logical palette. If a control enters a state where it no longer has a palette, it should clear the CTLFLG_HASPALETTE flag using the **VBSetControlFlags** function. This prevents the control from receiving VBM_GETPALETTE messages.

### Default Action

Returns NULL.

**See Also**

**VBSetControlFlags**

VBM_PALETTECHANGED

## VBM_GETPROPERTY

See Also

Requests that the control procedure supply the value of a property. This message will be sent only if the PF_fGetMsg flag is set in the propertys declaration. Note that if the PF_fGetData flag is set, Visual Basic reads the property value directly from the control structure.

Any attempt to access a property   including use of the Properties window, accessing the property from a Visual Basic statement, saving the form to disk, or a call to the **VBGetControlProperty** function   may result in this message being sent.

| Parameter | Description |
|---|---|
| *wp* | Index of the property within the controls property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lp* | Far pointer to the data item. If the PF_fPropArray flag is set, this parameter points to a DATASTRUCT structure. The **VBGetControlProperty** function performs any appropriate conversion after data is returned. |

### Comments

In the case of HSZ data types, Visual Basic assumes that you are supplying the handle of a temporary string, so it proceeds to destroy the string and free the memory after the string has been used. Therefore, use **VBCreateHsz** to create a new string that can be destroyed after use. By contrast, when the PF_fGetData flag is set, Visual Basic reads the appropriate HSZ in the programmer-defined structure and does not free the string space afterward, because it is assumed that the data needs to be maintained.

This difference between the uses of the PF_fGetMsg and PF_fGetData flags is consistent with the purpose of the VBM_GETPROPERTY message. This message is appropriate when the property setting is not maintained in memory, but needs to be determined on the fly.

### Return Value

Zero, if successful, or an error code.

### Default Action

Supplies the property value for standard properties.

**See Also**

**VBGetControlProperty**
VBM_CHECKPROPERTY
VBM_SETPROPERTY

## VBM_GETPROPERTYHSZ

See Also

Indicates that a property is about to be displayed in the Properties window. The control procedure should respond by specifying a text string to display in the Settings box. This message is sent only if the property declaration includes the PF_fGetHszMsg flag.

Setting the PF_fGetHszMsg flag (and responding to this message) should not be done for most properties. With simple data types, Visual Basic displays the value in the Properties window as you would expect. With DT_COLOR values, the value is displayed as a hexadecimal number. With Boolean values, the constant TRUE or FALSE is displayed. With DT_ENUM (enumerated values), you specify the list of strings to choose from in the **npszEnumList** field in the property information table.

Responding to this message is useful when neither the standard data type representations nor an enumerated list suffices. For example, the Picture property of a picture control displays only general information   (bitmap), (icon), (none)   rather than a data representation. However, with DT_PICTURE properties, this action is implemented for you.

Such a string cannot be edited in the Properties window; the user must use a pop-up dialog box to set the property. The control procedure must respond to the VBM_INITPROPPOPUP message to support this technique.

| Parameter | Description |
|---|---|
| *wp* | Index of the property within the controls property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lp* | Pointer to a handle of an HSZ string. You should create an HSZ string and copy the handle to the location pointed to by *lp*. Note that Visual Basic destroys the HSZ strings after HSZ is used. |

### Return Value

Zero, if successful, or an error code.

### Default Action

Processes the message for all standard properties and for custom properties that do not set the PF_fGetHszMsg flag. For custom properties that do set this flag, the control should return a value in *lp*.

**See Also**

VBM_INITPROPPOPUP

## VBM_HELP [2.0]

Generated for three different conditions:

- Help on currently selected property.
- Help on currently selected event.
- Help on currently selected control.

**Help on Currently Selected Property**

When the Property list or the edit box of the Properties window has the focus and the user presses F1, Visual Basic sends a VBM_HELP message to the control.

| Parameter | Description |
|---|---|
| *wp* | The low byte is VBHELP_PROP. The high byte is the index of the property within the controls property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lp* | A far pointer to the controls model structure. |

**Help on Currently Selected Event**

When the Procedure box in the Code window has the focus and the user presses F1, Visual Basic version 2.0 sends a VBM_HELP message to the control.

| Parameter | Description |
|---|---|
| *wp* | The low byte is VBHELP_EVT. The high byte is the index of the event within the controls event information table. The first event is indexed as 0, the second as 1, and so on. |
| *lp* | A far pointer to the controls model structure. |

**Help on Currently Selected Control**

When a control is selected on the form, selected in the Controls list of the Properties window, or when focus is on the controls icon in the Toolbox, pressing F1 sends a VBM_HELP message to the control.

| Parameter | Description |
|---|---|
| *wp* | VBHELP_CTL. |
| *lp* | A far pointer to the controls model structure. |

### Comments

A custom control typically calls the **WinHelp** function to display a specified topic in the custom controls own Help file.

**Note** Custom control should not use the control handle passed with this message, since VBM_HELP is a model message, and thus the control handle is NULL when the control procedure receives this message.

### Default Action

Attempts to find a Help topic based on the specified property, event, or control. If a Help topic is found, WinHelp displays the topic from the VB.HLP Help file. If a Help topic cannot be found, WinHelp displays the default for the type of topic:

| Topic type | Default topic |
|---|---|
| Property | Table of Contents for properties. |
| Event | Table of Contents for events. |
| Object | Table of Contents for objects. |

If the custom control contains a property or event that is the same name as a standard property or event, the custom control must process the VBM_HELP message correctly. Otherwise, WinHelp displays the standard property or event instead of the custom controls property or event.

## VBM_HITTEST [2.0]

Sent to the graphical control any time the mouse is moved over the rectangle that surrounds the graphic. The custom control must return one of the defined HT_ codes.

| Parameter | Description |
|---|---|
| *wp* | Unused. |
| *lp* | **LPHITTEST** Points to the **HITTEST** data structure that identifies the *x*- and *y*-coordinates of the cursor, and the **RECT** values of the control. Coordinate values and **RECT** values are relative to the parent window. |

If desired, the return value may differ when Visual Basic version 2.0 is running in design mode versus run mode. Typically, only HT_ON and HT_MISS are returned when in run mode, since calculating a hit or miss is easier, and therefore more efficient. Some controls, for example the standard Visual Basic Line control, always return HT_MISS in run mode.

In design mode, however, the custom control may return a greater number of messages to aid users when manipulating and designing forms.

### Return Value

One of the following codes should be returned directly from the controls window procedure.

| Code | Meaning |
|---|---|
| HT_ON | Point lies on the control. Typically reserved for only the perimeter of a polygon or circle. |
| HT_SOLID_NEAR | Point lies within a few (typically 4) pixels of an area returning HT_ON, and the point lies in an area which is painted solidly. For example, the interior of a solid circle. |
| HT_PATTERN_NEAR | Point lies within a few pixels of an area returning HT_ON, but the point lies in an area which contains a semitransparent hatch pattern. For example, the interior of a circle containing a FillStyle > 1, and a BackStyle of Transparent. |
| HT_HOLLOW_NEAR | Point lies within a few pixels of an area returning HT_ON, but the point lies in an unpainted (totally transparent) area. For example, the interior of an unfilled circle. |
| HT_SOLID | Just like HT_SOLID_NEAR, but does not lie within a few pixels of an area returning HT_ON. |
| HT_PATTERN | Just like HT_PATTERN_NEAR, but does not lie within a few pixels of an area returning HT_ON. |
| HT_HOLLOW | Just like HT_HOLLOW_NEAR, but does not lie within a few pixels of an area returning HT_ON. |
| HT_MISS | Point does not lie on, in, or near (a few pixels from) the control. |

### Default Action

Always returns HT_ON.

## VBM_INITIALIZE

See Also　　　Example

Sent after the control structure has been allocated but before anything else is loaded or allocated, including the window structure. This message is received only if the MODEL_fInitMsg flag is set in the control model.

This message is useful to respond to if you have pre-HWND properties that you want to initialize.

| Parameter | Description |
|---|---|
| *wp* | Unused |
| *lp* | Unused |

**See Also**

VBM_CREATED

**Example**

```
case VBM_INITIALIZE:
  lpmyctl = (LPMYCTL)VBDerefControl(hctl);
  lpmyctl->borderstyle = 1;
  break;
```

## VBM_INITPROPPOPUP

See Also

Sent after a property has been selected in the Properties window. The response to this message determines how values are displayed and accessed in the Settings box.

Depending on the return value, you can have the property set through direct editing (no special handling), with a standard drop-down list, or with a pop-up window.

| Parameter | Description |
|---|---|
| *wp* | Index of the property within the controls property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lp* | Low word: window handle of the list box that Visual Basic supplies. High word: unused. |

### Comments

To display a pop-up dialog box, first create a window as described in the table for the Other return value. When the window gets a WM_SHOWWINDOW message, it should hide itself and then post a message to itself that, when received, causes it to display the dialog box.

If you create a custom list of properties, you need to set the index item for the list when responding to this message. For example, you might define a DT_HSZ property array. Heres how you might respond to the VBM_INITPROPPOPUP message:

```
SendMessage(LOWORD(lp), LB_ADDSTRING, 0, (LPARAM)(LPSTR)"Canada");
SendMessage(LOWORD(lp), LB_ADDSTRING, 0, (LPARAM)(LPSTR)"Mexico");
SendMessage(LOWORD(lp), LB_ADDSTRING, 0, (LPARAM)(LPSTR)"USA");
err = (ERR)VBGetControlProperty(hctl, IPROP_MYCONTROL_COUNTRY,
                        &hszCountry);
if (err) return err;

lpCountry = VBLockHsz(hszCountry);
SendMessage(LOWORD(lp), LB_SELECTSTRING, -1, lpCountry);
VBUnlockHsz(hszCountry);
VBDestroyHsz(hszCountry);
return lp;
```

### Return Value

You can either return NULL, the value provided in the *lp* parameter, or a value identifying your own pop-up window. In the latter two cases, the text in the Settings box cannot be edited unless the PF_fEditable flag is set for this property.

| Return value | Description |
|---|---|
| NULL | No special handling. Visual Basic developer edits value directly in the Settings box, and no list is provided. |
| LOWORD(*lp*) | If you return this value supplied in *lp,* Visual Basic provides a list box and enables the down arrow to the right of the Settings box. Before returning this value, you should first have added items to the list by sending LB_ADDSTRING (sorted) or LB_INSERTSTRING (nonsorted) messages to the list box. Visual Basic ensures that the item selected is assigned to the property value. |
| Other | You can also supply the handle of a pop-up window you create in response to this message. Visual Basic displays the window when the developer clicks the arrow to the right of the Settings box, which displays ellipses (...). The window is responsible for setting the property value. |

**Default Action**

Creates a custom dialog box for the standard icon and DT_PICTURE properties, uses the list box for DT_ENUM and DT_BOOL properties, and returns NULL otherwise.

**See Also**

**VBDialogBoxParam**

VBM_GETPROPERTYHSZ

## VBM_ISMNEMONIC [2.0]

See Also

The VBM_ISMNEMONIC message is sent to the control when a mnemonic is entered. When ALT+*char* is pressed for a character which does not match a menu item or a mnemonic on the form, this message is sent to all controls on the form, until a control returns TRUE.

| Parameter | Description |
|---|---|
| *wp* | ASCII code of character that is the mnemonic. |
| *lp* | Unused. |

**Note** VBM_ISMNEMONIC and VBM_WANTSPECIALKEY are dynamic. That is, each time a special key or a potential mnemonic is pressed, one of these messages is sent. Therefore, a control can respond differently depending upon its state.

### Comments

The VBM_ISMNEMONIC message, paired with the new argument to the VBM_MNEMONIC message, allows a control to have an arbitrary number of mnemonics.

### Return Value

Return TRUE if *wp* is a mnemonic recognized by this control. Otherwise, return FALSE.

### Default Action

If the MODEL_fMnemonic flag is set, then Visual Basic checks the controls Text or Caption property, via WM_GETTEXT, to determine if it contains an &*char* pair, returning TRUE if found. Otherwise, FALSE is returned.

**See Also**

VBM_MNEMONIC

VBM_WANTSPECIALKEY

## VBM_LINKENUMFORMATS [2.0]

Example

Sent when either the client or the server needs to know the type of DDE data formats that the custom control can send or receive. The client or server may send this message any number of times in order to enumerate all the controls data formats.

| Parameter | Description |
|---|---|
| *wp* | SUPPLIESDATAFORMAT or ACCEPTSDATAFORMAT. |
| *lp* | The low word contains the current enumerated format, starting from 0. Each time the client or server requests additional data format values, the low word of *lp* is incremented by one. |

### Comments

If *wp* = SUPPLIESDATAFORMAT, then the control enumerates the data formats it supplies for a DDE conversation. If *wp* = ACCEPTSDATAFORMAT, then the control enumerates the data formats it accepts.

### Return Value

The return value is the data format that can be received or sent. The returned formats should be in order of preference   the most preferred data format is the first format returned.

Returning NULL stops the enumeration, meaning that you have returned all the data formats that you support.

### Default Action

Returns NULL.

**Example**

```
case VBM_ENUMFORMATS:
  switch (LOWORD(lp)) {
    case 0:
      return CF_BITMAP;
    case 1:
      return CF_METAFILEPICT;
    case 2:
      return CF_TEXT;
    default:
      return NULL;
  }
  break;
```

## VBM_LINKGETDATA [2.0]

See Also Example

Requests from the control through DDE.

| Parameter | Description |
|---|---|
| *wp* | An integer value that corresponds to the format of the data, such as CF_TEXT or CF_BITMAP. |
| *lp* | A far pointer to a VBLINKDATA structure that the control fills in. The control must allocate global memory for the **hData** field, and set the **cb** field to the length of data returned. |

### Comments

The data placed in *lp* should be in the format specified by *wp*.

### Return Value

The return value should be one of the following:

| Return value | Description |
|---|---|
| LINK_DATA_OK | Data successfully placed in *lp*. |
| LINK_DATA_OOM | Unable to allocate data   out of memory. |
| LINK_DATA_FORMATBAD | Unable to create data in the requested format. |

### Default Action

Returns LINK_DATA_FORMATBAD.

**See Also**

VBM_LINKENUMFORMATS
VBM_LINKGETITEMNAME
VBM_LINKSETDATA

**Example**

```
case VBM_LINKGETDATA:
  {LPSZ lpsz;

  if (wp != CF_TEXT)
    return LINK_DATA_FORMATBAD;

  ((LPDDEDATA)lp)->cb =
      lstrlen(VBDerefHsz(LpcircDEREF(hctl)->hszCaption)) + 1;

  ((LPDDEDATA)lp)->hData =
    GlobalAlloc(GMEM_DDESHARE | GMEM_MOVEABLE,
      ((LPDDEDATA)lp)->cb + 1);

  if (!((LPDDEDATA)lp)->hData)
    return LINK_DATA_OOM;          // Out of memory

  lpsz = (LPSZ)GlobalLock(((LPDDEDATA)lp)->hData);
  lstrcpy(lpsz, VBDerefHsz(LpcircDEREF(hctl)->hszCaption));
  GlobalUnlock(((LPDDEDATA)lp)->hData);
  return LINK_DATA_OK;}
```

## VBM_LINKGETITEMNAME [2.0]

See Also

Sent when either the client or the server needs to know the name of the item to which it is linked. In certain cases, specifying the name of the control as the DDE link topic is not sufficient. For example, if you wanted to provide a link to a specific cell in a spreadsheet-like control, you would have to specify the cell coordinates as the DDE link item.

**Note** A custom control cannot support multiple links; therefore, only one item name is valid per DDE conversation.

| Parameter | Description |
|---|---|
| *wp* | LINKSRCASK or LINKSRCTELL. |
| *lp* | A far pointer to a string. If *wp* = LINKSRCASK, *lp* contains the DDE item name string. |

### Return Value

If *wp* = LINKSRCASK, the control should return TRUE if it wants to use the item name, contained in *lp,* in a DDE conversation. Otherwise, the control should return FALSE.

If *wp* = LINKSRCTELL, the control should copy the DDE item name to *lp* and return TRUE. The maximum length of the item name is MAXLINKITEMNAME.

### Default Action

If *wp* = LINKSRCASK, Visual Basic returns the comparison of the DDE item name string and the Name property string.

If *wp* = LINKSRCTELL, the Name property of the control is copied to *lp*. If the control is a control array element, the index is appended to the control name, such as *ctrlName*(*n*).

**See Also**

VBM_LINKENUMFORMATS
VBM_LINKGETDATA
VBM_LINKSETDATA

## VBM_LINKSETDATA [2.0]

See Also Example

Sent when the control receives data by the control through DDE.

| Parameter | Description |
|---|---|
| *wp* | An integer value that corresponds to the format of the data, such as CF_TEXT or CF_BITMAP. |
| *lp* | A far pointer to the received VBLINKDATA structure. |

### Comments

The data placed in *lp* is in the format specified by *wp*.

### Return Value

The return value should be one of the following:

| Return value | Description |
|---|---|
| LINK_DATA_OK | Data successfully extracted from *lp*. |
| LINK_DATA_OOM | Unable to allocate data out of memory |
| LINK_DATA_FORMATBAD | Unable to use data in the specified format. |
| LINK_DATA_SETFAILED | Any other type of error not specified above. |

### Default Action

Returns LINK_DATA_FORMATBAD.

**See Also**

VBM_LINKENUMFORMATS
VBM_LINKGETITEMNAME
VBM_LINKGETDATA

**Example**

```
case VBM_LINKSETDATA:
  {
  ERR   err;
  LPSZ  lpsz;

  // Assure that the Clipboard format is of type text.
  if (wp != CF_TEXT)
    return LINK_DATA_FORMATBAD;

  // If this were CF_BITMAP, CF_DIB or CF_METAFILEPICT, hData
  // would indicate a block of memory that contained a handle to
  // the data, requiring a double dereference.

  // Set the linked data as the control's Caption.
  lpsz = (LPSZ)GlobalLock(((LPDDEDATA)lp)->hData);
  err = VBSetControlProperty(hctl, IPROP_CIRCLE_CAPTION,
        (LONG)lpsz);
  GlobalUnlock(((LPDDEDATA)lp)->hData);

  if (err)
    return LINK_DATA_SETFAILED;

  return LINK_DATA_OK;
  }
```

## VBM_LOADED

Sent to a control after all the controls on a form have been loaded, and after a control has been created as a dynamic array element with a **Load** statement at run time.

This message is sent only if the MODEL_fLoadMsg flag is set in the control model.

| Parameter | Description |
|---|---|
| *wp* | Unused |
| *lp* | Unused |

### Return Value

Zero, if successful, or an error code.

## VBM_LOADPROPERTY

See Also

Requests that the value of a property be read from a disk file that is in a binary format. Visual Basic sends this message during the loading of a form, but only if the property is declared with the PF_fSaveMsg or PF_fLoadMsgOnly flags set. If only the PF_fSaveData flag is set, Visual Basic does not send this message; it reads the data from disk directly and then sets the value of the property.

You should respond by using the **VBReadFormFile** function, which reads data from disk that was previously saved for that property in response to a VBM_SAVEPROPERTY message.

| Parameter | Description |
|---|---|
| *wp* | Index of the property within the controls property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lp* | The handle to a form file (HFORMFILE) value to use with the **VBReadFormFile** function. |

### Comments

See notes on the **VBReadFormFile** function for a discussion of when the PF_fSaveMsg flag is useful.

### Return Value

Zero, if successful, or an error code.

### Default Action

Handles loading of standard properties.

**See Also**

**VBReadFormFile**
VBM_LOADTEXTPROPERTY
VBM_SAVEPROPERTY

## VBM_LOADTEXTPROPERTY [2.0]

See Also Example

Requests that the value of a property be read from a disk file that is in an ASCII text format. Visual Basic sends this message during the loading of a form, but only if the property is declared with the PF_fSaveMsg or PF_fLoadMsgOnly flag set. If the PF_fSaveData flag is set, Visual Basic does not send this message; it reads the data from disk directly and then sets the value of the property.

| Parameter | Description |
|---|---|
| *wp* | Index of the property within the controls property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lp* | A far pointer to a null-terminated string that contains the saved data. The format of the data is defined by the control. |

### Return Value

Zero, if successful, or an error code.

### Default Action

Handles loading of standard properties.

**See Also**

VBM_SAVETEXTPROPERTY

**Example**

```
case VBM_LOADTEXTPROPERTY:
  switch (wp) {
    case IPROP_CIRCLE_CIRCLESHAPE:
      lpcirc = (LPCIRC)VBDerefControl(hctl);
      // Retrieve the text value of CircleShape.
      if (!lstrcmpi((LPSTR)lp, "Circle"))
        lpcirc->CircleShape = TRUE;
      else
        lpcirc->CircleShape = FALSE;
  ...}
```

## VBM_METHOD

Indicates that one of several predefined methods has been used with the control in a Visual Basic statement. These methods are **AddItem**, **Clear**, **Drag**, **LinkSend**, **Move**, **Refresh**, **RemoveItem**, and **ZOrder**. By responding to this message, you can add support for the **AddItem**, **Clear**, and **RemoveItem** methods, which are not supported by default. The **Drag**, **LinkSend**, **Move**, **Refresh**, and **ZOrder** methods are supported by default, but you can respond to this message to support any of these in a customized way.

**Note** The standard methods **LinkExecute**, **LinkPoke**, **LinkRequest**, and **SetFocus** are also automatically supported for controls. However, no notification is given to the control for these methods through the VBM_METHOD message.

You can choose to respond to the VBM_METHOD message for as many methods as you want, and accept default behavior for the rest. If you want to accept the default behavior, however, make sure that you call **VBDefControlProc**.

| Parameter | Description |
|---|---|
| *wp* | Index of the method:<br>METH_ADDITEM<br>METH_CLEAR [2.0]<br>METH_DRAG<br>METH_LINKSEND [2.0]<br>METH_MOVE<br>METH_REMOVEITEM<br>METH_REFRESH<br>METH_ZORDER [2.0] |
| *lp* | A far pointer to an array of long integers containing the method arguments, if any. The first argument, *cArgs*, gives the count of arguments including *cArgs* itself. Thus, if the arguments were *cArgs*, *hszItem*, and *index*, *cArgs* would be set to three. The argument lists are described below. |

### Comments

The arguments to each method are shown below. All arguments are numeric, except for *hszItem*, which is an HSZ handle to a string containing the new item.

| Method | Argument List |
|---|---|
| **AddItem** | cArgs, hszItem, index |
| **Clear** | No arguments: *lp* = NULL |
| **Drag** | *cArgs* [, *cmd*] |
| **LinkSend** | No arguments: *lp* = NULL |
| **Move** | *cArgs*, *left* [, *top* [, *width* [, *height*]]] |
| **Refresh** | No arguments: *lp* = NULL |
| **RemoveItem** | *cArgs*, *index* |
| **ZOrder** | cArgs, position:<br>position = 0 (bring to front)<br>position = 1 (send to back) |

### Return Value

Zero, if successful, or an error code.

### Default Action

In the case of **AddItem**, **RemoveItem**, and **Clear**, the default action is to generate a run-time error, since these methods are not supported by default. For the other methods, the default action is listed below.

| Method | Default action |
|---|---|
| **Drag** | Initiate drag mode, cancel, or drop the control. |
| **LinkSend** | Transfer data to client application in DDE conversation. |
| **Move** | Move (and size) the control as indicated by arguments. |
| **Refresh** | Invalidate and update window. |
| **ZOrder** | Change z-order of control: bring to front, send to back. |

## VBM_MNEMONIC

See Also

Indicates that the control has just received the focus because the user typed an access key, called a *mnemonic key* in Windows programming. A mnemonic key corresponds to the letter in the standard Caption property preceded by an ampersand (&).

Within a dialog box, support of mnemonic keys is handled by the dialog manager. Within a Visual Basic application, the controls determine how to interpret mnemonic keys.

Visual Basic automatically moves the focus in response to the appropriate control in response to a mnemonic key. To implement any further action (such as firing the Click event), you need to respond to this message.

| Parameter | Description |
|---|---|
| *wp* | Character that activated mnemonic. |
| *lp* | Unused. |

**Note** The *wp* value is unused in Visual Basic version 1.0 (equal to 0), but set to the mnemonic character value in Visual Basic version 2.0.

**See Also**

VBM_ISMNEMONIC

VBM_WANTSPECIALKEY

## VBM_PAINT [2.0]

Sent to graphical controls when Windows makes a request to repaint a portion of the control.

| Parameter | Description |
|---|---|
| *wp* | An hDC value that identifies the device context. |
| *lp* | An **LPRECT** that points to the **RECT** data structure that identifies the controls coordinates. |

### Comment

The hDC must be restored to its original state any resources newly selected into the hDC (brushes, pens, and other objects) must be deselected, and deleted if necessary.

## VBM_PAINTMULTISEL [2.0]

Sent when the control becomes part of a multiple selection. If nondefault behavior is desired, the control can respond by painting the appropriate gray multiple selection handles around the graphic.

| Parameter | Description |
|---|---|
| *wp* | An hDC value that identifies the device context. |
| *lp* | An **LPRECT** that points to the **RECT** data structure identifying the area to paint the multiple selection handles. |

### Default Action

Paints eight gray multiple selection handles in the default positions around the control.

## VBM_PAINTOUTLINE [2.0]

Sent when the control is being moved. The control can respond by drawing the XOR representation of the control. This message is sent in design mode only.

| Parameter | Description |
|---|---|
| *wp* | An hDC value that identifies the device context. |
| *lp* | An **LPRECT** that points to the **RECT** data structure identifying the area to paint the outline. |

### Comment

When the VBM_PAINTOUTLINE message is sent during the creation of a control, *hctl* is NULL. When moving or sizing an existing control, *hctl* is a valid value.

### Default Action

Paints a default gray outline sizing rectangle.

## VBM_PALETTECHANGED [2.0]

See Also

Sent to the control when Windows makes a request to select a logical palette. This message is only sent to palette-aware controls.

| Parameter | Description |
|---|---|
| *wp* | A **BOOL** value. TRUE if selecting palette for background; FALSE if selecting palette for foreground. |
| *lp* | Unused. |

### Comment

The control should select its palette into its own hDC using *wp* as the *fPalBack* parameter to the Windows API **SelectPalette** function. The *wp* value determines foreground or background selection.

The control should select and realize a palette to the foreground only in response to VBM_PALETTECHANGED when *wp* is FALSE. All other palette realizations, even for painting, should be to the background.

The control usually invalidates itself, so the control should defer any painting until it receives the WM_PAINT message.

### Return Value

The control should return TRUE if and only if it realizes a palette, and the Windows API **RealizePalette** function returns a nonzero value. Otherwise, return FALSE.

### Default Action

Gets the controls palette via VBM_GETPALETTE and realizes the palette, using the controls hDC, and then invalidates the control.

**See Also**

**VBSetControlFlags**

VBM_GETPALETTE

## VBM_PASTE

See Also

Notifies the control that it should accept information (or a paste link) contained in the Clipboard. Receiving this message indicates that the control has previously agreed, in response to a VBM_QPASTEOK message, to proceed with a paste operation. This message is sent at design time only.

If a control is present in the Clipboard and the Paste command is chosen, Visual Basic does not send this message; it simply handles the paste operation itself. This message is sent to the control when data in some other Clipboard format is available. This includes data prepared by some application outside of Visual Basic. For example, if the currently selected control is a picture box, it can accept any data in the standard bitmap format; this data may have been generated by a Paint program.

It is up to the control procedure to decide exactly how to apply the information available in the Clipboard.

| **Parameter** | **Description** |
|---|---|
| *wp* | PT_PASTE, if the Paste command was chosen, or PT_PASTELINK, if the Paste Link command was chosen. |
| *lp* | Unused. |

### Return Value

Zero, if successful, or an error code.

**See Also**

VBM_COPY
VBM_QPASTEOK

## VBM_QPASTEOK

See Also

Sent when the user opens the Edit menu in design mode. Visual Basic sends this message to ask the control if it should enable the Paste or Paste Link command. This message is sent at design time only.

If a control is present in the Clipboard and the Paste command is chosen, Visual Basic does not send this message. Instead, it enables the Paste command directly. The purpose of this message is to see if the currently selected control accepts data in some other Clipboard format, such as text, bitmap, or metafile. Your code can respond by using the Windows API **IsClipboardFormatAvailable** function to determine if the appropriate type of data is available.

If you dont want your control to be able to paste in data in one of these standard Clipboard formats, you can ignore this message.

| Parameter | Description |
|---|---|
| *wp* | PT_PASTE, if the Paste command should be enabled, or PT_PASTELINK, if the Paste Link command should be enabled. |
| *lp* | Unused. |

### Comments

If you are writing a control that is compatible with Visual Basic version 2.0 and that uses DDE, and the *wp* parameter is PT_PASTELINK, you should call the **VBPasteLinkOK** function. Use the return value of the function as the return value of this message.

### Return Value

TRUE if the control can accept information currently in the Clipboard; FALSE otherwise.

### Default Action

Returns FALSE.

**See Also**

**VBPasteLinkOK**

VBM_COPY

VBM_PASTE

## VBM_SAVEPROPERTY

See Also

Requests that the value of a property be written to disk. Visual Basic sends this message as a form is being saved, but only if the property is declared with the PF_fSaveMsg flag set. By contrast, if the PF_fSaveData flag is set, Visual Basic does not send this message; instead, it gets the value of the property and then writes the data to disk directly.

| Parameter | Description |
|---|---|
| *wp* | Index of the property within the property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lp* | The handle to a form file (HFORMFILE) to use with the **VBReadFormFile** function. |

### Comments

See notes on the **VBReadFormFile** function for a discussion of when the PF_fSaveMsg flag is useful.

### Return Value

Zero, if successful, or an error code.

### Default Action

Responds by saving the property value for any property that can receive this message.

**See Also**

**VBWriteFormFile**

VBM_LOADPROPERTY

VBM_SAVETEXTPROPERTY

## VBM_SAVETEXTPROPERTY [2.0]

See Also Example

Allows a control to save its property values in an ASCII file format. This message is sent only if the PF_fSaveMsg flag is set for the property.

| Parameter | Description |
|---|---|
| *wp* | Index of the property within the property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lp* | A far pointer to an HSZ that contains the text representation of the property. |

### Comments

The *lp* parameter is a pointer to a location to receive an HSZ. If you want to respond to this message, you must first create a new HSZ, which represents the textual representation of the value of the property indicated by *wp*. Then you set **lp* to the HSZ value:

```
*(LPSTR)lp = hsz;
```

After sending this message, Visual Basic checks the value pointed to by the *lp* parameter. If *(LPSTR)*lp* is NULL (meaning the control did not respond to the message), the VBM_SAVEPROPERTY message is sent. If *(LPSTR)*lp* is not NULL, then the data is copied to the ASCII file and the VBM_SAVEPROPERTY message is not sent.

### Return Value

Zero, if successful, or error code.

### Default Action

Handles saving standard property values.

**See Also**

VBM_LOADTEXTPROPERTY
VBM_SAVEPROPERTY

**Example**

```
case VBM_SAVETEXTPROPERTY:
  switch (wp) {
    case IPROP_CIRCLE_CIRCLESHAPE:
      // Save the value of CircleShape as text.
      lpcirc = (LPCIRC)VBDerefControl(hctl);
      if (lpcirc->CircleShape)
        lpszText = Circle;
      else
        lpszText = Oval;

      hsz = VBCreateHsz((_segment)hctl, lpszText);
      if (hsz)
        *(LPSTR)lp = hsz;
      else
        return 7;// Out of memory

      return 0;
      ...}
```

## VBM_SELECTED [2.0]

Indicates that the user has selected the control at design time. The message is sent before the Properties window is updated. This allows the control to modify property values, such as the enumerated strings for a property defined as DT_ENUM.

| Parameter | Description |
|---|---|
| *wp* | Specifies whether the property to be displayed in the Properties window can be part of a multiple selection. Refer to the PF_fNoMultiSelect property flag in Chapter 13. |
| *lp* | Unused. |

## VBM_SETPROPERTY

See Also Example

Requests that the control procedure set a property to the value provided. This message is sent only if the PF_fSetMsg flag is set in the propertys declaration. Note that if the PF_fSetData flag is set instead, Visual Basic assigns the new property value directly to the control structure. If both flags are set, Visual Basic first assigns the value to the control structure and then sends this message.

| Parameter | Description |
|---|---|
| *wp* | Index of the property in the controls property information table. The first property is indexed as 0, the second as 1, and so on. |
| *lp* | Data to copy into the property setting. Format of the data depends on the type. Because *lp* is declared as **LONG**, it should be cast to the correct type before being used. In the case of floating-point data, this requires that you take the address of *lp* and cast it as a pointer to a four-byte floating-point number, then use indirection to get the data. (See the following example.)<br>In the case of DT_HSZ properties, *lp* is a far pointer to null-terminated string data. In the case of DT_HLSTR properties, *lp* is a far pointer to an HLSTR, which must be copied before you use it. In the case of DT_PICTURE properties, *lp* is an HPIC.<br>If the PF_fPropArray flag is set, this parameter points to a DATASTRUCT structure. |

### Comments

Any attempt to set a property   use of the Properties window to set a value, assigning a value to the property from a Visual Basic statement, loading the form from disk, or a call to the **VBSetControlProperty** function   may produce this message.

The **VBSetControlProperty** function performs any appropriate conversion before data is passed in this message.

**Note** If you allow the default processing to handle the VBM_SETPROPERTY message and the text that is passed with the WM_SETTEXT message is greater than 256 bytes, **VBDefControlProc** sets the 256th byte to NULL.

### Return Value

Zero, if successful, or an error code.

### Default Action

Responds by setting the property value for any standard property that can receive this message.

**See Also**

**VBSetControlProperty**
VBM_CHECKPROPERTY
VBM_GETPROPERTY

**Example**

```
case VBM_SETPROPERTY:
  switch (wp) {
    case IPROP_MYCTL_REALAMT:
      pmyctl->realAmt = *(float *)&lp;
      InvalidateRect(hwnd, NULL, TRUE);
      return 0;
    ...}
```

## VBM_WANTSPECIALKEY [2.0]

See Also

Sent to the control when a virtual key generates a WM_KEYUP or WM_KEYDOWN message. This message is sent only for keys which are normally trapped by VB, including: VK_ESCAPE, VK_CANCEL, VK_EXECUTE, VK_RETURN, VK_TAB, VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN.

| Parameter | Description |
|---|---|
| *wp* | Virtual key code. |
| *lp* | Unused. |

**Note** VBM_ISMNEMONIC and VBM_WANTSPECIALKEY are dynamic, that is each time a special key or a potential mnemonic is pressed, one of these messages is sent. Therefore, a control can respond differently depending upon its state.

### Comments

VBM_WANTSPECIALKEY allows controls to trap ENTER, ESC, TAB, and arrow keys when those keys would normally have some special meaning.

### Return Value

Returning TRUE prevents a key from taking normal meaning. For example, returning TRUE for VK_TAB will prevent the tab key from tabbing the focus away from that control.

### Default Action

Returns TRUE if *wp* is an arrow key and the controls model has MODEL_fArrows set. Otherwise, default processing returns FALSE.

**See Also**

VBM_ISMNEMONIC
VBM_MNEMONIC

## VBN_ Messages

| Visual Basic message | Corresponding Windows message |
|---|---|
| VBN_CHARTOITEM [2.0] | WM_CHARTOITEM |
| VBN_COMMAND | WM_COMMAND |
| VBN_COMPAREITEM | WM_COMPAREITEM |
| VBN_CTLCOLOR | WM_CTLCOLOR |
| VBN_DELETEITEM | WM_DELETEITEM |
| VBN_DRAWITEM | WM_DRAWITEM |
| VBN_HSCROLL | WM_HSCROLL |
| VBN_MEASUREITEM | WM_MEASUREITEM |
| VBN_PARENTNOTIFY | WM_PARENTNOTIFY |
| VBN_VKEYTOITEM [2.0] | WM_VKEYTOITEM |
| VBN_VSCROLL | WM_VSCROLL |

### Comments

Each of these messages indicates that the control has sent a corresponding WM_ message to its parent. Generally, these messages are most useful when you are creating a subclass of an existing Windows control. A Windows control often sends notification to its parent, and the VBN mechanism provides a convenient way to handle this behavior.

For example, a button might send a WM_COMMAND message to its parent. Within Visual Basic, the parent window is actually the controls container; this is typically the form, but it also might be a picture box or frame. In any case, the default behavior for all forms and controls is to reflect back the message as a VBN message. So if a control sends a WM_COMMAND message to a parent, it gets back a VBN_COMMAND message. If a control sends a WM_CTLCOLOR message, it gets back a VBN_CTLCOLOR message. The control procedure can then process the VBN message as appropriate.

See Chapter 10, Subclassing a Windows Control, for more explanation and examples.

### Return Value

Defined by the corresponding Windows message.

### Default Action

For VBN_CTLCOLOR, the default control procedure sets the display context and returns a background brush, according to the values of the standard BackColor and ForeColor properties.

## Visual Basic Standard Properties

Visual Basic provides 44 standard properties. Each of these is represented by a constant that points to a PROPINFO structure for the property. The default processing routine, **VBDefControlProc** handles the appropriate message processing for each of these properties.

Note that all the size and position properties (Left, Top, Width, Height) are written to or read from disk together when the Left property is saved or loaded. Therefore, make sure you include Left as a property if you want your control to contain any size or position information.

Similarly, all the font properties (FontName, FontBold, FontItalic, FontStrike, FontUnder, FontSize) are written to or read from disk together when the FontName property is saved or loaded. Therefore, make sure you include FontName as a property if you want your control to contain any font information.

Properties that are specific to a version level of Visual Basic are denoted by the version number inside brackets. For example, version 3.0 is denoted by [3.0].

| | | |
|---|---|---|
| Align [2.0] | FontName | LinkTimeout [2.0] |
| BackColor | FontSize | LinkTopic [2.0] |
| BorderStyle (off) | FontStrike | MousePointer |
| BorderStyle (on) | FontUnder | Name [2.0] |
| Caption | ForeColor | None [2.0] |
| ClipControls [2.0] | Height | Parent |
| CtlName | HelpContextID [2.0] | TabIndex |
| DataChanged [3.0] | hWnd | TabStop |
| DataField [3.0] | ImeMode [2.0] | Tag |
| DataSource [3.0] | Index | Text |
| DragIcon | Last [2.0] | Top |
| DragMode | Left | TopNoRun [2.0] |
| Enabled | LeftNoRun [2.0] | Visible |
| FontBold | LinkItem [2.0] | Width |
| FontItalic | LinkMode [2.0] | |

### Align [2.0]

PPROPINFO_STD_ALIGN

The Align property, when set to *align to top* or *align to bottom*, forces the width of the control to be the same width as the form's ScaleWidth. The control's position is also aligned to the top or bottom of the form.

If multiple controls are aligned to the same position on a form, the controls are stacked on top of each other.

The WM_SIZE and WM_MOVE messages are still sent to the control, but the size and position of the control cant be changed − any changes to the size and position are ignored.

**BackColor**

PPROPINFO_STD_BACKCOLOR
You must send a WM_CTLCOLOR message to get a brush that contains the background color.

**BorderStyle (off)**

PPROPINFO_STD_BORDERSTYLEOFF

Assumes the window style defined in the control's MODEL structure does NOT contains the WS_BORDER flag.

**BorderStyle (on)**

PROPINFO_STD_BORDERSTYLEON

Assumes the window style defined in the control's MODEL structure contains the WS_BORDER flag.

**Caption**

PPROPINFO_STD_CAPTION

Requires that you respond to WM_SETTEXT, WM_GETTEXT, and WM_GETTEXTLENGTH messages.

**ClipControls [2.0]**

PPROPINFO_STD_CLIPCONTROLS

Sets or clears the window style flag WS_CLIPCHILDREN, based on the value of this property at design-time. This property is valid only for controls that set the MODEL_fChildrenOk flag.

**CtlName**

PPROPINFO_STD_CTLNAME

This property is required, and should be placed first in the list. Replaced with PPROPINFO_STD_NAME in Visual Basic version 2.0.

**DataChanged [3.0]**

PPROPINFO_STD_DATACHANGED

Indicates that the data in the bound control has been changed by some process other than getting the data from the current record.

**DataField [3.0]**

PPROPINFO_STD_DATAFIELD

Allows your control to bind to a field in a database. This property requires that you implement the DataSource property.

**DataSource [3.0]**

PPROPINFO_STD_DATASOURCE

Allows your control to bind to a specific data control. This is the only property needed to initiate binding.

This property requires that you add the MODEL_fLoadMsg flag to the controls model structure.

**DragIcon**

PPROPINFO_STD_DRAGICON

**DragMode**

PPROPINFO_STD_DRAGMODE

**Enabled**

PPROPINFO_STD_ENABLED

**FontBold**

PPROPINFO_STD_FONTBOLD

**FontItalic**

PPROPINFO_STD_FONTITALIC

**FontName**

PPROPINFO_STD_FONTNAME

Requires that you respond to WM_SETFONT and WM_GETFONT messages. Responding to these messages provides the support necessary for all the font properties.

All font properties are saved and loaded together with FontName.

**FontSize**

PPROPINFO_STD_FONTSIZE

**FontStrike**

PPROPINFO_STD_FONTSTRIKE

**FontUnder**

PPROPINFO_STD_FONTUNDER

**ForeColor**

PPROPINFO_STD_FORECOLOR
You must send a WM_CTLCOLOR message to use this property.

**Height**

PPROPINFO_STD_HEIGHT

**HelpContextID [2.0]**

PPROPINFO_STD_HELPCONTEXTID

The HelpContextID property is the user-defined help context ID. This context ID corresponds to a topic in the user's help file.

**hWnd**

PPROPINFO_STD_HWND

The hWnd property is set to the control's window handle. The property is read only, and not available at run time.

For graphical controls, this property is not valid.

**ImeMode [2.0]**

PPROPINFO_STD_IMEMODE

The ImeMode allows the control to recognize the DBCS mode. If a control is loaded with this property in the domestic version of the product, it is ignored. This allows you to create one custom control that runs under non-DBCS (domestic) as well as DBCS (international) versions.

**Index**

PPROPINFO_STD_INDEX
This property is required, and should be placed second in the list.

## Last [2.0]

PPROPINFO_STD_LAST

Indicates the maximum value of a standard property (IPROP_STD_*property*). Therefore, this property also indicates the lowest value of a standard property (PPROPINFO_STD_*property*) in the control's PropertyList array. A value less than PPROPINFO_STD_LAST is a non-standard property.

This property is not a real property, but only a marker. Use it to test if a property is a standard or user-defined property. For example:

```
usCurrProp = lpmodel->npproplist[i];
if (usCurrProp < PPROPINFO_STD_LAST) {
   // non-standard property processing
} else {
   // standard property processing
}
```

**Left**

PPROPINFO_STD_LEFT
Left, Top, Width, and Height are all saved and loaded together with Left.

**LeftNoRun [2.0]**

PPROPINFO_STD_LEFTNORUN
Use this property instead of Left for an invisible control.

**LinkItem   [2.0]**

PPROPINFO_STD_LINKITEM

The LinkItem property corresponds to the *ItemName* value in standard DDE syntax, *AppName*|*TopicName*|*ItemName*.

**LinkMode   [2.0]**

PPROPINFO_STD_LINKMODE

The LinkMode property determines the type of link used for a DDE conversation and activates the connection.

**LinkTimeout [2.0]**

PPROPINFO_STD_LINKTIMEOUT

The LinkTimeout property determines the amount of time a control waits for a response to a DDE message.

**LinkTopic [2.0]**

PPROPINFO_STD_LINKTOPIC
The LinkTopic property determines the application name and topic of a DDE conversation.

**MousePointer**

PPROPINFO_STD_MOUSEPOINTER

**Name [2.0]**

PPROPINFO_STD_NAME

This property is required, and should be placed first in the list. Replaces PPROPINFO_STD_CTLNAME in Visual Basic version 1.0.

**None [2.0]**

PPROPINFO_STD_NONE

The None property allows you to remove a property by replacing it with a placeholder that does nothing. This allows applications to use different versions of a control.

**Parent**

PPROPINFO_STD_PARENT
The control's parent. All controls are required to have this property.

**TabIndex**

PPROPINFO_STD_TABINDEX
Should be supported if MODEL_fFocusOk or MODEL_fMnemonic flag is set.

**TabStop**

Should be supported if the MODEL_fFocusOk flag is set.

**Tag**

PPROPINFO_STD_TAG

**Text**

PPROPINFO_STD_TEXT
This property is the same as Caption, except for the property name.

**Top**

PPROPINFO_STD_TOP

**TopNoRun [2.0]**

PPROPINFO_STD_TOP
Use this property instead of Top for an invisible control.

**Visible**

PPROPINFO_STD_VISIBLE

**Width**

PPROPINFO_STD_WIDTH

# Data Structures

This chapter provides a summary of each data structure used in custom control code. Understanding these structures can be useful in writing general DLL routines as well as in writing custom controls. Features that are specific to a version level of Visual Basic are denoted by the version number inside brackets. For example, version 3.0 are denoted by [3.0].

## The MODEL Structure

| Field name | Description |
|---|---|
| **usVersion** | An unsigned integer specifying the version of Visual Basic for which the control was developed. Typically initialized with VB_VERSION, a constant defined in VBAPI.H. |
| **fl** | MODEL flags (listed in the next section). |
| **pctlproc** | The address of the control procedure. |
| **fsClassStyle** | The window class styles for the control. These class styles all have a CS prefix, and they supplement the class styles in the superclass, if one exists. |
| **flWndStyle** | Default window styles for the control. |
| **cbCtlExtra** | Size of the programmer-defined structure, if any, to be allocated as part of each control structure. Set to 0 if there is no programmer-defined structure. |
| **idBmpPalette** | Starting resource ID for bitmaps that provide the control's Toolbox icon. Position and meaning of the bitmaps are:<br>*idBmpPalette* VGA, up position<br>*idBmpPalette*+1 VGA, down position<br>*idBmpPalette*+3 Monochrome, up position<br>*idBmpPalette*+6 EGA, up position<br>The down position bitmaps for EGA and monochrome displays are not required − Visual Basic inverts the up position bitmap when it is selected. |
| **npszDefCtlName** | Default control name. Visual Basic uses this string to assign default values to Name (CtlName for Visual Basic version 1.0). For example, the first three Circle controls are named Circ1, Circ2, and Circ3. Segment assumed is that of the control model. |
| **npszClassName** | Visual Basic class name. This name can be used in a Visual Basic **If...TypeOf** statement to recognize the control's type.<br>The class name is displayed in the Object box of the Properties window. Segment assumed is that of the control model. |
| **npszParentClassName** | The class name of the subclassed control, if any, or NULL if there is no subclassing. Segment assumed is that of the control model. |
| **npproplist** | A near pointer to the properties information table, which is an array of pointers to PROPINFO structures. Segment assumed is that of the control model. |
| **npeventlist** | A near pointer to the event information table, which is an array of pointers to EVENTINFO structures. Segment assumed |

| | |
|---|---|
| | is that of the control model. |
| **nDefProp** | Index of default property in Properties window. |
| **nDefEvent** | Index of default event in Code window. |
| **nValueProp** [2.0] | Index of property that is the default property value for the control. Setting this field to -1 indicates no default property value.<br>If a control is created, with the name property set to Circle1, and the default value property is set to Caption, then the following code:<br>`Circle1 = "hello"`<br>means<br>`Circle1.Caption = "hello"`<br>Future versions of the custom control API may assume that a property with the name "Value" is the default property. If your control includes a Value property, it should be designated as the nValueProp property. If your control does not include a Value property, you are free to use any other property in your model. Any future version which defaults to using the Value property will instead use the property that you designate in the **nValueProp** field if a Value property is not defined for the control |
| **usCtlVersion** [3.0] | Indentifies the current version of the custom control. Refer to Technical Note 2: *Custom Control Version Management*, TN002.TXT, in the CDK directory. |

## MODEL Flags

| Flag value | Description |
|---|---|
| MODEL_fArrows | Send keyboard messages for arrow keys to the control. If this flag is off, arrow keys are used to move between controls. See VBM_WANTSPECIAL_KEY. |
| MODEL_fChildrenOk | Enable the application developer to draw other controls inside of this control at design time. This lets the control function as a container of other controls, as can picture boxes and frames. |
| MODEL_fDesInteract | Enable the control to get right-mouse-button messages at design time: WM_RBUTTONDOWN, WM_RBUTTONDBLCLK, WM_RBUTTONUP.<br>By processing these messages at design time, the control procedure can provide special mechanisms for manipulating the control and its properties. |
| MODEL_fFocusOk | Enable the control to get the focus at run time. |
| MODEL_fGraphical [2.0] | Indicate control is a graphical control. |
| MODEL_fInitMsg | Enable the control to get VBM_INITIALIZE messages. |
| MODEL_fInvisAtRun [2.0] | Specify that the control is invisible at run time. Visual Basic automatically handles the representation of the control on the form |

| | |
|---|---|
| | at design time − it uses the same icon that appears in the Toolbox. |
| MODEL_fLoadMsg | Enable the control to get VBM_CREATED and VBM_LOADED messages. This flag is required for bound controls and DDE-enabled controls. |
| MODEL_fMnemonic | Respond to the control's mnemonic key (access key) by moving the focus to the control and sending a VBM_MNEMONIC message. The control must support WM_GETTEXT and WM_GETTEXTLENGTH messages, or the standard Caption or Text property in order to set this flag. |

## The PROPINFO Structure

| Field name | Description |
|---|---|
| **npszName** | The name of the property as it appears in Visual Basic statements and in the Properties window. This name must conform to Visual Basic variable naming conventions. |
| **fl** | Property flags (listed in the next two sections). |
| **offsetData** | The offset of the field in the programmer-defined structure where the property value is stored. This field can be ignored only if neither the PF_fGetData nor the PF_fSetData flag is set for the property.<br>The OFFSETIN macro, described in Chapter 6 of the *Control Development Guide,* "Adding a Custom Property," is useful for initializing this field. |
| **infodata** | If this field is nonzero, and the data type is DT_BOOL, DT_SHORT, or DT_ENUM, the field supports packing of data within the programmer-defined structure. The data must range from 0 to 15, so that it can be packed within 4 bits or fewer. This field can be ignored only if neither the PF_fGetData nor the PF_fSetData flag is set for the property.<br>The highest 4 bits of infodata specify a bit mask, which should be binary − 0001, 0011, 0111, or 1111, depending on the size of the bit field. The lowest 4 bits give the shift count, which specifies what bit the field starts in. See Chapter 11 of the *Control Development Guide,* "Other Programming Topics," for examples. |
| **dataDefault** | If the PF_fDefVal flag is set, this field should contain the most common value, so that when the control is saved to disk, the property is skipped if it contains the dataDefault value. Then, when the control is loaded, the property is set to the dataDefault value (unless PF_fNoInitDef is also set). |
| **npszEnumList** | For enumerated properties (DT_ENUM), this field is a pointer to a string providing list box items for the Properties window. Items are separated by embedded nulls. Two nulls in a row terminate the string. Setting to NULL indicates no items. |
| **enumMax** | For enumerated properties (DT_ENUM), this field indicates the maximum legal value. Setting to 0 turns off automatic validation. |

## Property Data Type Flags

Only one of the following flags can be selected for each property. The selected flag is then logically OR'ed with flags listed in the next section.

| Flag value | Resulting data type of property |
|---|---|
| DT_BOOL | Boolean. This is a short integer, though it can be packed into as little as 1 bit. All incoming and outgoing nonzero values are converted to -1 during a set or get operation. |
| DT_COLOR | A long integer that holds a color value. This value is an RGB value, or (if the most significant bit is set) the lowest 3 bytes contain a system-color number defined in WINDOWS.H.<br>Visual Basic automatically validates numbers input for validity as color values. The value is displayed in the Properties window using hexadecimal radix. |
| DT_ENUM | A short integer holding an enumerated value (0 −255). Selection of this data type (as opposed to DT_SHORT) enables the **npszEnumList** and **enumMax** fields, described in the previous section. The data can be packed into a bit field from 1 to 4 bits wide. |
| DT_HLSTR [2.0] | String. The string data should be stored as an HLSTR type. DT_HLSTR property values are string descriptors representing strings of bytes that can contain embedded NULL characters. This is the principle advantage over DT_HSZ strings which are terminated by a NULL character.<br>HLSTR strings that you allocate must be freed in response to WM_NCDESTROY.<br>Refer to Chapter 9 of the *Control Development Guide,* "Handling Strings and Fonts," for more details on HLSTR strings. |
| DT_HSZ | String. The string data should be stored as an HSZ type. If the PF_fSetData flag is set, allocation and freeing of the HSZ string are handled by Visual Basic.<br>HSZ strings that you allocate must be freed in response to WM_NCDESTROY.<br>Refer to Chapter 9 of the *Control Development Guide,* "Handling Strings and Fonts," for more details on HSZ strings. |
| DT_LONG | 32-bit signed integer. |
| DT_OBJECT | Points to an iDispatch interface. Refer to Technical Note 1: *Support for DT_OBJECT Properties*, TN001.TXT, in the CDK directory. |
| DT_PICTURE | Picture structure. The picture should be stored as an HPIC handle. If the PF_fSetData flag is set, Visual Basic allocates HPIC handles as needed. The HPIC must be freed in response to WM_NCDESTROY. |
| DT_REAL | 4-byte real. |
| DT_SHORT | 16-bit signed integer. The data can be packed into a bit field from 1 to 4 bits wide. |
| DT_XPOS | Long integer expressing an X coordinate. The data for a get or set operation is expressed in terms of the container's scale. The data gets converted, if necessary, so that the custom control code uses units expressed in twips, with the origin (0,0) at the control's upper-left corner. Similar conversions are performed for the remaining types. |
| DT_XSIZE | Long integer expressing an X size in twips. Origin does not affect size conversions. |
| DT_YPOS | Long integer expressing a Y coordinate in twips. |
| DT_YSIZE | Long integer expressing a Y size in twips. |

## Other Property Flags

You can use any number of the following flags in the **fl** field of the PROPINFO structure.

**Important** When the PF_fPropArray flag is used, the following flags must also be used: PF_fSetMsg, PF_fGetMsg, and PF_fNoShow. The PF_fSaveData flag must not be used.

| **Flag value** | **Description** |
|---|---|
| PF_fDefVal | Causes Visual Basic to avoid saving and loading the property to disk when the value is equal to the dataDefault value in the PROPINFO structure. This flag affects only loading and saving, not default values of newly created controls.<br><br>During saving, the property value is not written to disk if equal to the dataDefault value. Then, during loading, if the property was not saved to disk, it is set to the dataDefault value. (The PF_fNoInitDef flag alters loading behavior.) |
| PF_fEditable | Enables the application developer to edit text in the Settings box directly, even if a list box or pop-up window is used. If neither is used, this flag has no effect. |
| PF_fGetData | Indicates that Visual Basic gets the property value by directly copying data from the programmer-defined structure. The **offsetData** and **infoData** fields of the PROPINFO table specify exactly where the data is located within the structure. |
| PF_fGetHszMsg | Causes Visual Basic to send a VBM_GETPROPERTYHSZ message to the control whenever the property value is to be displayed in the Properties window. If this flag is not used, then Visual Basic simply gets the property value and displays it in the Settings box. |
| PF_fGetMsg | Causes Visual Basic to send a VBM_GETPROPERTY message when the property value is requested. Either this flag or the PF_fGetData flag should be used. |
| PF_fLoadMsgOnly [2.0] | Causes Visual Basic to send VBM_LOADPROPERTY messages when the form is loaded from a file. These messages pass a handle to file location. It is then the responsibility of the control procedure to read from disk (using the **VBReadFormFile** function).<br><br>This property is similar to PF_fSaveMsg, except that a VBM_SAVEPROPERTY messages are not generated. This allows custom controls to be backward compatible by loading properties in previous versions that are no longer supported in newer ones. |
| PF_fLoadDataOnly [2.0] | Causes Visual Basic to load the property from a form file, but prevents the property from being written back out to the form file.<br><br>This property is similar to PF_fSaveData, except that properties are not written out to the form file. This allows custom controls to be compatible with Visual Basic version 1.0. |
| PF_fNoMultiSelect [2.0] | Specifies that any given custom property NOT be displayed in the Properties window when that control is part of a selected group. Currently, no properties of type DT_PICTURE are included in a multiple selection.<br><br>However, if PF_fNoMultiSelect is not present, a future version of Visual Basic may include a given property of type DT_PICTURE in a multiple selection. If you must guarantee that a property is never placed in a multiple selection (even if it is DT_PICTURE), then PF_fNoMultiSelect must be set. |
| PF_fNoInitDef | Prevents Visual Basic from setting the property to the dataDefault value during loading of the control (though you can still initialize the property by responding to messages). If PF_fDefVal is not set, PF_fNoInitDef has no effect. |

| | |
|---|---|
| PF_fNoRuntimeR [2.0] | Indicates that the property is write-only at run time. Any attempt to read the property generates a runtime error. When combined with PF_fNoRuntimeW, it indicates that the property is not available at all at run time (a design-time-only property), and references to it are flagged as Visual Basic compilation errors. |
| PF_fNoRuntimeW | Indicates that the property is read-only at run time. |
| PF_fNoShow | Prevents the property from appearing in the Properties window. |
| PF_fPreHwnd | Causes the property to be loaded before the control's window structure is created. A property should be pre-HWND if it affects a window style. The actual setting of the window style (that is, the style altered by the pre-HWND property) should be made in response to a WM_NCCREATE message. |
| PF_fPropArray | Specifies that the property is an array. When this flag is used, you must use a DATASTRUCT structure to implement all getting and setting of the property. The PF_fNoShow property must also be set. |
| PF_fSaveData | When the form is being saved to a file, Visual Basic saves the property value by getting its value and then writing the data to disk directly. Similarly, Visual Basic loads the property by reading its value directly from the disk and then setting it. |
| PF_fSaveMsg | Causes Visual Basic to send a VBM_SAVEPROPERTY message to the control whenever the form is being saved to a file, and a VBM_LOADPROPERTY message when the form is loaded from a file.<br><br>These messages pass a handle to a file. It is then the responsibility of the control procedure to read or write from disk (using the functions **VBReadFormFile** and **VBWriteFormFile**). |
| PF_fSetCheck | Causes Visual Basic to send a VBM_CHECKPROPERTY message before it sets the value of the property. This gives the custom control code a chance to check the validity of the property value.<br><br>If the control procedure returns a nonzero error code in response to this message, Visual Basic reports an error to the user and will not proceed with setting the property to the new value. |
| PF_fSetData | Indicates that Visual Basic sets the value of the property by placing data directly in the programmer-defined structure. |
| PF_fSetMsg | Causes Visual Basic to send a VBM_SETPROPERTY message when setting of the property is attempted. Either this flag or the PF_fSetData flag should be used. If both are used, then the data is transferred before the message is sent. |
| PF_fUpdateOnEdit | Causes the property value to be set every time a character is typed in the Settings box of the Properties window. If this flag is not used, the property is not set until change is committed.<br><br>Typically used for string properties like Caption and Text. |

## The EVENTINFO Structure

| Field name | Description |
|---|---|
| **npszName** | The name of the event as it appears in the Code window. |
| **cParms** | Number of arguments, not including the *Index* argument. |
| **cwParms** | Total number of words represented by the argument list. Since each argument is based as a 2-byte pointer to the data, this field is always twice the value of cParms. |
| **npParmTypes** | A pointer to a BYTE array that lists the type of each argument, in the order listed in the event procedure header. Each element of this array is an ET flag that indicates an argument type. For example, a pointer to a BYTE array is initialized as: `{ET_I2, ET_SD}` |
| **npszParmProf** | A null-terminated string which contains the procedure arguments as they are to appear in the event procedure header. This string should include commas to separate arguments, but no parentheses. The *Index* argument is not included; Visual Basic takes care of adding that argument when appropriate. This string must not exceed 180 bytes − for example: `"I As Integer, FirstName As String"` |
| **fl** | This field can contain the EF_fNoUnload flag, or be set to 0. When this flag is set, it tells Visual Basic not to allow unloading of the current form or any control on the form, while the corresponding event procedure is being fired. The EF_fNoUnload flag is useful for events that must assume that nothing is unloaded. (The standard Paint event is an example.) |

## Argument Type Flags

| Value | Description |
|---|---|
| ET_I2 | 16-bit signed integer |
| ET_I4 | 32-bit signed integer |
| ET_R4 | 4-byte real |
| ET_R8 | 8-byte real |
| ET_CY | 8-byte currency |
| ET_HLSTR | String (new name for ET_SD) |

## The PIC Structure

The PIC structure type is defined in VBAPI.H as follows:

```
typedef struct tagPIC
{
   BYTE  picType;
   union {
      struct   {
         HBITMAP     hbitmap;   // Bitmap
         HPALETTE hpal;         // Accompanying palette
      } bmp;
      struct   {
         HANDLE      hmeta;        // Metafile
         int         xExt, yExt;   // Extent
      } wmf;
      struct   {
         HICON    hicon;           // Icon
      } icon;
   } picData;
   BYTE  picReserved[4];
}
PIC;
```

The use of each field is described below.

| Field name | Description |
|---|---|
| **picType** | Enumerated type specifying the format of picture. Can be set to any of the following:<br>PICTYPE_NONE<br>PICTYPE_BITMAP<br>PICTYPE_METAFILE<br>PICTYPE_ICON |
| **picData.bmp.hbitmap** | Handle to a bitmap. |
| **picData.bmp.hpal** [2.0] | Handle to a palette. |
| **picData.wmf.hmeta** | Handle to a metafile. |
| **picData.wmf.xExt** | Width in twips of the area to contain the picture. |
| **picData.wmf.yExt** | Height in twips of the area to contain the picture. |
| **picData.icon.hicon** | Handle to an icon. |

## The DATASTRUCT Structure (Property Arrays)

The DATASTRUCT data structure type is defined in VBAPI.H as follows:

```
typedef struct
{
   LONG  data;                // Data for Get and Set
   USHORT   cindex;           // Number of indexes
                              // (currently always 1)
   struct
   {
      USHORT   datatype;      // Type of nth index (Always
                              // DT_SHORT)
      LONG  data;             // Value of nth index
   } index [1];               // Currently, only 1-dim
                              // arrays supported
} DATASTRUCT;
```

The use of each field is described below. If you call **VBSetControlProperty** or **VBGetControlProperty** to manipulate a property array, you need to first declare a DATASTRUCT structure and initialize the fields to appropriate values. When you respond to VBM_SETPROPERTY and VBM_GETPROPERTY, the structure is passed to you and you can assume that it is already filled with acceptable values. When responding to VBM_GETPROPERTY, you place the property value in the first field:

| Field name | Description |
|---|---|
| **data** | Data to be transferred. In the case of numeric and DT_PICTURE types, the field is used the same way whether getting or setting the property. The field contains the actual data to be transferred − or an HPIC handle, in the case of a DT_PICTURE type. (This field should be initialized to zero before a call to **VBGetControlProperty**; the data returned is placed in this field as a result of the call.) |

In the case of strings, the meaning of the field changes between the "get" and the "set" case. The field should contain an HSZ handle as a result of getting the property value; it contains a pointer to the null-terminated string data when being used to set a property value.

**cindex** The number of indexes. Only one-dimensional property arrays are supported in this version of Visual Basic, so this field should always contain the value 1.

**index[0].datatype** The data type of the index, as specified by a DT flag. This field should always contain DT_SHORT for this version.

| | |
|---|---|
| **index[0].data** | Index of the element involved in the data transfer. |

## The DRAGINFO Structure

This structure is passed in the *lp* parameter of VBM_DRAGOVER and VBM_DRAGDROP messages. The DRAGINFO data structure type is defined in VBAPI.H as follows:

```
typedef struct tagDRAGINFO
{
   HCTL  hctl;
   POINT pt;
   USHORT   state;   // Enter, Over, Exit; only used for VBM_DRAGOVER
} DRAGINFO;
```

| Field name | Description |
|---|---|
| **hctl** | Handle to the control structure. |
| **pt** | A POINT structure containing the mouse coordinates. |
| **state** | Used with the VBM_DRAGOVER message only: this field is an unsigned integer indicating the state of the drag. The value of this field can be one of the following constants:<br>DRAG_STATE_ENTER<br>DRAG_STATE_EXIT<br>DRAG_STATE_OVER |

## The VBLINKDATA Structure   [2.0]

The VBLINKDATA structure is passed in the *lp* parameter of VBM_LINKGETDATA and VBM_LINKSETDATA messages. The VBLINKDATA data structure is defined in VBAPI.H as follows:

```
typedef struct tagVBLINKDATA
   {
   WORD  wReserved;      // reserved
   DWORD cb;             // size of data
   HANDLE   hData;       // handle to data
   DWORD dwReserved;     // reserved
   } VBLINKDATA;

typedef VBLINKDATA FAR *LPLINKDATA;
```

| Field name | Description |
|---|---|
| **wReserved** | Reserved. |
| **cb** | The size of the data in the buffer pointed to by the *hData* handle. |
| **hData** | Handle to a global buffer containing the linked data. |
| **dwReserved** | Reserved. |

## The MODELINFO Structure [2.0]

The MODELINFO structure is used by the **VBGetModelInfo** function. The MODELINFO data structure is defined in VBAPI.H as follows:

```
typedef struct tagMODELINFO
   {
   USHORT         usVersion;       // VB version used by
                                   // control
   LPMODEL FAR    *lplpmodel;// pointer to null-
                                   // terminated
   } MODELINFO;                    // list of LPMODELS

typedef MODELINFO FAR *LPMODELINFO;
```

| Field name | Description |
|---|---|
| **usVersion** | An unsigned integer value that represents the Visual Basic version of the control. |
| **lplpmodel** | Pointer to null-terminated array of pointers to MODEL structures. |

## The DATAACCESS Structure   [3.0]

The DATAACCESS structure is used by the Visual Basic set of data binding messages. The DATAACCESS data structure is defined in VBAPI.H as follows:

```
typedef struct
{
   USHORT usVersion;       // VB version of structure filled
                                // in when structure is created
   SHORT sAction;           // on VBM_DATA_GET/SET specifies
                                // what to get/set
                                // on VBM_DATA_AVAILABLE/REQUEST
                                // tells why
   HCTL hctlData;           // the data control providing   data
   HCTL hctlBound;          // the bound control receiving data
   HSZ hszDataField;       // the name of the field to get
 value of
   SHORT sDataFieldIndex; // the field index used when
                     // hszDataField is NULL
   HLSTR   hlstrBookMark;           // used when getting multirow data
   FSHORT fs;            // Bitfield structure
   USHORT usDataType;    // the property datatype to convert data to
   LONG     lData;       // the data
   ULONG   ulChunkOffset;           // the offset to start at for GetChunk
   ULONG   ulChunkNumBytes;   // the number of bytes for GetChunk
} DATAACCESS, FAR * LPDATAACCESS;
```

| Field name | Description |
|---|---|
| **usVersion** | An unsigned integer value that represents the Visual Basic version of the DATAACCESS structure. Whichever control creates the structure fills in this value. |
| **sAction** | An integer value that represents the action to perform when the VBM_DATA_GET or VBM_DATA_SET messages are sent, or the action completed when the VBM_DATA_AVAILABLE or VBM_DATA_REQUEST messages are received. You also specify this value when you send the VBM_DATA_METHOD message.<br>Refer to the specific message for a description of all **sAction** values. |
| **hctlData** | Handle to the data controls control structure. This is the data control that the bound control references via the DataSource property. Use this value when responding to a VBM_DATA_AVAILABLE or VBM_DATA_REQUEST message. However, dont store the **hctlData** value and attempt to use it later for processing other messages. |
| **hctlBound** | Handle to the bound controls control structure. |
| **hszDataField** | Handle to the name of the field. Set to NULL if referencing the field by **sDataFieldIndex**. |
| **sDataFieldIndex** | An integer value that represents the index of the field. Only used if **hszDataField** is NULL. |
| **hlstrBookMark** | Handle to a bookmark. This is typically used when traversing the records in a recordset using the VBM_DATA_GET message. |
| **fs** | Bitfield structure that contains the following flags: |

**DA_fNull** -- set to TRUE if the field value is null. For numeric values, this allows you differentiate between a 0 value and a null value.

**DA_fBOF** -- set to TRUE if the current record position is before the first record in a recordset. When traversing a recordset via the VBM_DATA_GET message (*sAction* set to DATA_BOOKMARK), this flag allows you to determine the beginning of the recordset.

**DA_fEOF** -- set to TRUE if the current record position is after the last record in the recordset. When traversing a recordset via the VBM_DATA_GET message (*sAction* set to DATA_BOOKMARK), this flag allows you to determine the end of the recordset.

**usDataType** Requested data type of the **lData** value when retrieving a field value from the data control via VBM_GET_MESSAGE. The data control attempts to coerce the field value into the request data type if possible.

When sending the **lData** value to the data control via VBM_SET_MESSAGE, **usDataType** represents the actual data type of the value.

The only valid data types for **usDataType** are property data types (for example, DT_HSZ).

**lData** Data value. The value received via VBM_GET_DATA, or the value sent via VBM_SET_DATA.

**ulChunkOffset** Unsigned long that represents the offset to use when using DATA_FIELDCHUNK with the VBM_DATA_GET message.

**ulChunkNumBytes** Unsigned long that represents the size of the chunk when using DATA_FIELDCHUNK with the VBM_DATA_GET message.

## Standard Events

Visual Basic provides 18 standard events. When you include a standard event in your custom control code, you need only declare it in the event information table. If you choose, you can fire the event yourself, but this is normally unnecessary. The default control procedure, **VBDefControlProc** fires standard events in response to specific messages.

---

**Note** If the standard Click, MouseDown, or MouseUp event is included, the default control procedure automatically captures or releases the mouse.

---

Properties that are specific to a version level of Visual Basic are denoted by the version number inside brackets. For example, version 2.0 is denoted by [2.0].

| | | |
|---|---|---|
| Click | KeyPress | MouseUp |
| DblClick | KeyUp | LinkClose [2.0] |
| DragDrop | Last [2.0] | LinkError [2.0] |
| DragOver | LostFocus | LinkNotify [2.0] |
| GotFocus | MouseDown | LinkOpen [2.0] |
| KeyDown | MouseMove | None [2.0] |

### Click

PEVENTINFO_STD_CLICK

Fired when the control has captured the mouse, a WM_LBUTTONUP message is received, and the button is released over the control. The standard MouseDown and MouseUp events occur before this event.

**DblClick**

PEVENTINFO_STD_DBLCLICK
Similar to Click, but fired when a WM_LBUTTONDBLCLK message is received.

**DragDrop**

PEVENTINFO_STD_DRAGDROP
Fired after a VBM_DRAGDROP message is received.

**DragOver**

PEVENTINFO_STD_DRAGOVER
Fired after a VBM_DRAGOVER message is received.

**GotFocus**

PEVENTINFO_STD_GOTFOCUS

After receiving a WM_GOTFOCUS message, the default message processor posts a VBM_FIREEVENT message. This mechanism delays firing of the event until other pending messages are processed.

**KeyDown**

PEVENTINFO_STD_KEYDOWN
Fired when a WM_KEYDOWN or a WM_SYSKEYDOWN message is received.

**KeyPress**

PEVENTINFO_STD_KEYPRESS
Fired when a WM_CHAR message is received.

**KeyUp**

PEVENTINFO_STD_KEYUP
Fired when either a WM_KEYUP or a WM_SYSKEYUP message is received.

**Last [2.0]**

Indicates the maximum value of a standard event (IEVENT_STD_*event*). Therefore, this event also indicates the lowest value of a standard event (PEVENTINFO_STD_*event*) in the control's EventList array. A value less than PEVENTINFO_STD_LAST is a non-standard event.

This event is not a real event, but only a marker. Use it to test if an event is a standard or user-defined event. For example:

```
usCurrEvent = lpmodel->npeventlist[i];
if (usCurrEvent < PEVENTINFO_STD_LAST) {
   // non-standard event processing
} else {
   // standard event processing
}
```

**LostFocus**

PEVENTINFO_STD_LOSTFOCUS

Fired when a WM_LOSTFOCUS message is received. The same delayed-processing mechanism is used as for GotFocus.

**MouseDown**

PEVENTINFO_STD_MOUSEDOWN

Fired if any BUTTONDOWN (right, left, or middle button) message is received. The mouse is captured as a result of this event.

**MouseMove**

PEVENTINFO_STD_MOUSEMOVE
Fired when a WM_MOUSEMOVE message is received.

**MouseUp**

PEVENTINFO_STD_MOUSEUP

Fired if any BUTTONDOWN (right, left, or middle button) message is received. The mouse capture is released as a result of this event.

**LinkClose [2.0]**

PEVENTINFO_STD_LINKCLOSE
Fired when a DDE conversation terminates.

**LinkError [2.0]**

PEVENTINFO_STD_LINKERROR
Fired when there is an error during a DDE conversation.

**LinkNotify [2.0]**

PEVENTINFO_STD_LINKNOTIFY
Fired when you have a DDE notify link and the data in the server has changed.

**LinkOpen [2.0]**

PEVENTINFO_STD_LINKOPEN
Fired when a DDE conversation is being initiated.

**None [2.0]**

The None event allows you to remove a event by replacing it with a placeholder that does nothing. This allows applications to use different versions of a control.