

## WinWidgets version 2.0

The WinWidgets are a set of custom controls for use with C/C++, Visual Basic and other Windows development tools. For an introduction to the WinWidgets, see [Welcome to the WinWidgets](#). For step-by-step instructions from installation through execution, see the [How do I...](#) guide. For detailed information about each tool, press the buttons in the Contents section below.

There are also Quick Reference buttons located in the toolbar to provide fast access to definitions and constants in C and C++. Within Visual Basic, press the F1 key while editing a property for context-sensitive help. For more information on using the documentation, see [Using the Manual](#).

Before using the WinWidgets, be sure to read the [License and Copyright](#) information.

---

---

### Contents



**How do I...?**



**DataEngine**



**Button Controls**



**ComboBox**



**Edit Control**



**Grid**



**ListBox**



**Static Controls**



**ToolBar**



**Frequently asked questions**

## Welcome to WinWidgets Version 2.0!

Welcome to WinWidgets v2.0, which now supports Visual Basic, C and C++.

With vVersion 2.0, we have adopted all-electronic documentation, so if you're wondering, "Where's my manual?" the answer is "It's right here!" This help file contains everything the printed documentation did plus a slew of extras. To get the most out of this documentation, see [Using the Manual](#).

To print all or part of this document from electronic form, choose **File, Print Manual...**. Select from the list of chapters and chapter sections those you want to have on paper. Then press **Print**. You may cancel the process at any time by pressing **Cancel**.

For a quick introduction to the WinWidgets, see the [Overview](#). The features new to Version 2.0 and the changes from previous versions are described in the [Version Notes](#) section.

If you would like help using the WinWidgets, see the [How do I...](#) guide for step-by-step instructions from installation through execution.

To get started, press one of the buttons below:

- Table of Contents
- Overview of the WinWidgets
- Using the Manual
- Version 2.0 Notes
- "How do I...?" Guide

# Version Notes

## Version 2.01

### New Features/Bug Fixes

#### Discontiguous selection in the grid

HGrid now supports the selection of discontiguous ranges of rows in WholeRows mode. When the control key is depressed, additional rows or ranges can be selected by clicking or dragging with the mouse without deselecting previously selected rows. See the Whole Rows attribute documentation in the help file for a description of the new API for getting and setting discontiguous selections through code.

Discontiguous selection is now the default for Whole Rows mode in the Grid.

#### Null data support in all controls

Explicit support for NULL data has been added to the Edit Control, CheckBox and Grid. The List and ComboBox already support NULL data by allowing no items to be selected (by setting the current selection to -1).

The format string for the edit control can now be extended with an additional semicolon to include a programmer-defined string that can be displayed when the data is NULL. The default is an empty string. For example, the format string "###-####;No Value" displays a telephone number when the data is valid and displays the string "No Value" when the data is NULL. The control will not leave the NULL state until the user enters valid data or the programmer resets the data.

The CheckBox displays a gray square when the data is NULL.

Any cell in the grid can be set to the NULL state and will format its data according to the NULL data formats of its child controls. The grid, itself cannot be put into a NULL state in the sense that there must always be at least one cell selected.

	<u>Setting the NULL State</u>	<u>Testing for the NULL State</u>
CheckBox	SetData with NULL pointer	GetData returns NULL
Edit Control	SetData with NULL pointer	GetData returns NULL
Grid Cell	HGSetCellData with NULL pointer	HGGetCellData return NULL

Grid Buffer Procedures and the NULL state:

NULL Cells are tracked by extending the record structure to include an array of WORDs for each field in the record. This array can be accessed in a buffer procedure or HGridBuffer-derived object member function by offsetting the lpData pointer by the size of the record data. Individual elements of the WORD array should be bitwise OR'ed with HGCF\_NULL to set the appropriate cells to NULL.

Two examples follow:

#### C Example:

A RECTSTRUCT is a programmer-defined record structure. For further information, see HGrid Record Structures.

```

BUFFERPROC MyBufferProc (HWND hwndGrid,
                        WORD wAction,
                        LONG lRecNum,
                        RECSTRUCT FAR *lpRecData )
{
    int i, iFields;
    WORD FAR *lpFlags;

    iFields =(int)SendMessage(hwndGrid, HGM_GETCOLCOUNT, 0, 0);

    switch(wAction)
    {
    case HGB_RETRIEVE: //Set the entire record to NULL state
        lpFlags = (LPBYTE)lpRecData + sizeof(RECSTRUCT);
        for (i=0; i<iFields, I++)
            lpFlags[i] |= HGCF_NULL;
        break;
    }
}

```

#### C++ Example:

```

BOOL CMyGridBuffer::OnRetrieve(LONG lRecNum, LPVOID lpData)
{
    int iFields = m_pGrid->GetColCount();;
    RECSTRUCT FAR *lpRecData = (RECSTRUCT FAR *)lpData;
    WORD FAR *lpFlags = (LPBYTE)lpRecData + sizeof(RECSTRUCT);

    //Set the entire record to NULL state
    for (int i=0; i<iFields; i++)
        lpFlags[i] |= HGCF_NULL;
}

```

### **C++ front end is now compiled for MSVC 1.5/MFC 2.5**

#### **Dropdown box width can be set in the ComboBox**

Use HCM\_GETDROPWIDTH and HCM\_SETDROPHEIGHT. These can be called in response to the CBN\_DROPDOWN notification command.

#### **Annoying brackets in read-only field titles removed**

GRS files created prior to version 2.01 will still have brackets. To change these, simply load in HGEEDIT.EXE and resave. If you are one of the few who miss the brackets, you can add them to the column title through the API's, HGFM\_SETNAME.

### **Changing marker during HGN\_SELCHANGING/HGN\_SELEXTENDING bug fix**

Several users have been trying to validate cell data by responding to HGN\_SELCHANGING and setting the Marker to the SelectionExtent in case of invalid data, which should return selection to the invalid cell. The bug was that when selection was changed with the mouse, the range between the old Marker and the new Marker was being selected. This has been fixed.

### **Final hidden row copy bug fix**

When copying a whole row of data to the clipboard, it was impossible to paste back into a whole row if the last field was hidden. This has been fixed.

### **Quotes around copied fields removed from clipboard text**

#### **Pasting and the selection in the Grid**

Pasting now puts the selection anchor at the lower right corner of the affected range and the extent at the upper left, instead of vice-versa.

#### **Memory leaks with bound VBX controls**

The List, ComboBox and Grid no longer leak memory when bound to the data control. The problem only occurred in the List and ComboBox when Codes were used.

### **Documentation Changes**

For the latest documentation, consult the on-line manual.

The documentation neglects to mention that the source code and resources required to produce the demo application are contained in the self-extracting compressed file, HDEMOSRC.EXE. Simply type:

```
HDEMOSRC /d
```

at the DOS command prompt and the file will decompress.

### **Additional notes on validation procedures**

HEdit data validation callback procedures, described in the Properties section of the HEdit documentation, offer a way to perform application specific checking of values entered by the user over and above that provided by the Data Engine (eg. restricting numerical values to a range). Programmers wishing to make use of validation procedures should be aware, however, that using the multiple procedures in a single dialog can result in unexpected behavior.

A control's validation procedure is called when that control loses focus, which means that some other control already has focus. It is only when the callback returns TRUE that focus is returned to the original control. If the control that gains focus also has a validation procedure installed, this procedure will be called when the first control restores focus to itself, and so on ad infinitum... If you use multiple validation procedures in a single dialog, these procedures must be aware of the execution status of all other validation procedures and return FALSE if another validation procedure is currently executing.

### **Planned Product Releases**

Simple Software is currently re-developing C++ wrapper classes to provide compatibility with Borland's Object Windows Library (OWL) version 2.0. We are also adding support for ODBC in MFC via the CRecordSet class. Future controls include a Tabbed-Dialog control, Spin Control, Progress Meter, and a Full-Functioned SpreadSheet Control. Also OwnerDraw support will added to the List and ComboBox controls. We plan to release WinWidgets

version 3.0 in early June, 1994.

## **Version 2.0**

### **New!**

Version 2.0 of the WinWidgets adds a wide range of new features including Visual Basic support and C++ classes that are compatible with Borland's OWL and the Microsoft Foundation Classes.

With this version we are adopting all-electronic documentation, which may come as a shock to some. Nonetheless, we feel this decision is essential to the progress of our tools, and we hope that our new version and completely redesigned manual lead you to agree.

### **Compatibility Issues**

*Filter procedures* have been removed almost as quickly as they were added. After a little research, we discovered a better way to subclass individual windows that has been a part of the SDK since version 2.0. The basic procedure is to retrieve the control's window procedure using `GetWindowLong()` and replace it with a custom window procedure; c, call the original procedure for any unaltered messages. We have encapsulated this procedure in the `SubclassWW()` function. See [subclassing the WinWidgets](#).

Starting with this version of the Grid control, we make a distinction between records and rows, and between fields and columns. Records are numbered starting from zero at the beginning of the data table up to a maximum of 2.1 billion. Rows are numbered from zero at the beginning of the data buffer to a maximum of 32,676 at the end of the buffer. A row index of -1 refers to the row of column buttons along the top of the Grid.

The distinction between fields and columns is necessitated by the drag-and-drop feature of the Grid. If drag-and-drop is not enabled, no distinction is necessary. Fields are numbered according to the position of their data in the record. Columns are numbered in the order in which they are displayed. A column index of -1 refers to the column of row buttons along the left side of the Grid. Hidden and non-scrolling fields maintain their field and column indices.



### **Fixes**

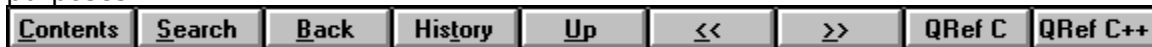
Yes, there have been a few...

## Using the Manual

To print all or part of this document from electronic form, choose **File, Print Manual...**. Select from the list of chapters and chapter sections those you want to have on paper. Then press **Print**. You may cancel the process at any time by pressing **Cancel**.

### A Few Conventions...

Throughout this manual you will see buttons like  and  that can be pressed for further information. Go ahead, try it! Other Help features to be aware of are the tool bar buttons --- press the buttons below for a description of their purposes:



Help also allows you to annotate the help file, add bookmarks, and to copy text from the manual to your application. To annotate a topic, choose **Edit, Annotate...**, type in your note and press **Save**. A green paperclip will appear at the beginning of the topic; press it to view or edit your annotation.

To mark your place in the manual, choose **Bookmark, Define...** and enter a bookmark name. The name will appear in the Bookmark menu and connect you with the marked topic until you delete the bookmark.

To copy text from the manual, choose **Edit, Copy...** In the edit box that pops up you can select all or part of the topic text and copy it to the clipboard by pressing **Copy**.

### Need a Guide?

If you are using the WinWidgets or custom controls for the first time, use the [How do I...](#) guide for step-by-step instructions from installation through execution of an application. To use the guide, press the button next to the **How do I...** label on the [Contents](#) page. The guide appears within a secondary window that can be left open as you select topics from the guide's instructions.

### Manual Organization

This manual documents the WinWidgets in three programming languages: - Visual Basic, C and C++. Despite their differences, all three languages treat the WinWidgets as objects that have *attributes* (e.g. Font, TextColor, Text, etc.), *methods* (things they can do, such as Clear, Update, etc.), and *events* (things that happen during their use, such as DbClick, SetFocus, etc.).

The manual is organized to exploit the language independence of objects:: it contains one chapter for each of the WinWidgets, and one topic in a chapter for each attribute, method and event. Only within topics are the distinctions in language implementation expanded.

**Note:** Standard Visual Basic properties are not documented in this manual; refer instead to the Visual Basic documentation.

For programmers using C and C++ who already know the language implementation, but need more details, we provide two Quick References. The Quick References offer brief descriptions of the window style bits, message constants, member functions, notification codes and window text for each control. They also connect these topics with the corresponding attributes, methods and events in the main documentation.

The manual begins with the Table of Contents, followed by this introduction, then the WinWidgets Guide. The rest of the documentation is divided by chapters, including one chapter for the [DataEngine](#) and one for each of the controls. The DataEngine chapter has topics for each class of data supported by the WinWidgets. The control chapters contain complete descriptions of each the control's attributes, methods, events and any other

topics unique to that tool. Each control's chapter begins with an introduction page that looks like this:

---

---

## **HButt, The WinWidgets Button Control**

- Attributes
- Events

HButt is an -all-purpose button control. Use it as a pushbutton, radiobutton, checkbox, or multiple-state button. It displays multiple lines of text and/or bitmaps and icons. It supports mnemonics and the default pushbutton style. It even plays sound resources. As a checkbox or radiobutton, HButt can be Hot-Linked to a data source that is automatically updated whenever the button changes state. All of the standard button types have a 3D appearance by default, but HButt can be easily tailored with custom images, styles and formatting.

### **Additional Topics**

- Using custom pictures and sounds
- Hot-Linking a button to your data
- Using HButt with the Visual Basic Data Control
- Displaying 256-color bitmaps

---

---

For information on the control's attributes (fonts, colors, text, etc.), press the button next to **Attributes**. This brings up the attribute list from which you select the attribute you want. The manual will move to the topic that discusses that attribute. You can use the same technique to learn about Methods and Events, although some of the controls may not have any methods or events.

### **Tips**

- To return to the control's introduction page from an attribute topic, press the **Up** button in the toolbar.
- To browse through a control's attributes in alphabetical order, go to the first attribute topic, then use the browse buttons (labeled << and >>) in the toolbar.

Takes you to the Table of Contents page, which has links to each of the WinWidgets' chapters, the DataEngine, and the **How do I...** guide.



Brings up the Search dialog, in which you can search for topics by keyword.

Takes you to the most recently viewed topic.

Brings up a list of the most recently viewed topics, from which you can select one to view.

Takes you "up" in the manual's topic hierarchy. From an attribute, method or event topic, it will take you to the control's introduction page,; from there it connects you to the Table of Contents.

Takes you to the preceding topic in alphabetical order. It is useful for browsing through a number of attributes, methods or events.

Takes you to the next topic in alphabetical order. It is useful for browsing through a number of attributes, methods or events.

Opens the C language Quick Reference in a secondary window.

Opens the C++ language Quick Reference in a secondary window.



## Overview

The WinWidgets toolset includes a complete set of custom controls and a central DataEngine that accelerate the development of business and science applications. The DataEngine is the brain behind the WinWidgets; it understands many types of data used in business and the sciences. Employing the DataEngine, the WinWidgets simplify the display of information and collection of user input. They also improve your program's usability with behavior and formatting options appropriate to each class of data.

The WinWidgets can be integrated with many popular design tools, including Microsoft Visual Basic, the AppStudio and ClassWizard, Borland's Resource Workshop and the SDK Dialog Editor. Within these tools, the WinWidgets provide interactive screen design, from the layout, text and background colors down to the data type and format specifications.

Often, the only coding required is the connection between the WinWidgets and your data. Our unique Hot-Linking feature can reduce your work to a single command for each control. When the user enters information into a Hot-Linked WinWidgets, the control updates the linked variable in your program automatically. Hot-Linking eliminates the need to poll the control every time its data is needed elsewhere; with Hot-Linking, program variables are always up-to-date.

Despite their advanced features, the WinWidgets remain extremely efficient by using an object-oriented class design to maximize the utility of code. This design is easily extendable, particularly when using the WinWidgets source code as a reference and baseline. Unlike other custom controls, the WinWidgets were written from scratch, not subclassed, so *all* of the source code is available. This makes the WinWidgets an excellent base for subclassing because there is no black box hiding their behavior.

The toolset contains the following controls:

HStat, the Static Control -- ideal for enhancing the appearance of screens and dialogs. It can be used as a simple panel or group box with easily selectable background colors and 3D border styles. HStat also displays bitmaps, icons and multiple lines of text. The WinWidgets design-time extensions allow easy customization of features, including:

- multiple lines of text with hard carriage returns and/or word wrapping
- left, right and center justification
- indented (3D) text
- text and background colors
- partially transparent bitmaps and transparent backgrounds

HButt, the Button Control -- the do-it-all button control, with pushbutton, default pushbutton, radiobutton, checkbox and three3-state styles. As a checkbox or radiobutton, the button's state can be Hot-Linked to a program variable. HButt can display text and/or icons and bitmaps. It allows complete customization of appearance and behavior through a simple design-time interface. Its features include:

- groupable pushbuttons that can be grouped to providewith radio button behavior
- "no-focus" buttons that are perfect for toolbars
- multiple lines of button text with left, right or center justification
- colored and/or indented (3D) text
- predefined 3D radio button and checkbox bitmaps, or looks created by youor you can create your own look
- simple, flexible alignment of text and bitmaps.

HEdit, the Edit Control -- an ideal tool for data entry, it employs the WinWidgets' built-in DataEngine and offers the WinWidgets' unique Hot-Linking feature. Combined, these abilities can reduce your interface code to a single line per control -- simply connect the edit control to your data, and it does the rest. Other features include:

- spreadsheet-style data formatting, such as "Mmm d, yyyy" or "\$#,##0.00"
- utilization of international settings for dates, times and numbers
- customizable validation routines
- 3D or plain appearance
- smart formatting with color - "\$#,##0.00 ;[RED](\$#,##0.00)"

HList, the List Control -- another improvement on the standard Windows control, HList employs the WinWidgets DataEngine and offers Hot-Linking to the current selection. It also allows non-displayed data, called *codes*, to be associated with each list item. Items can be selected and sorted by their displayed data or codes. Other features include:

- spreadsheet-style data formatting, such as "Mmm d, yyyy" or "\$#,##0.00"
- add arrays of items added with a single command
- 3D or plain appearance
- single, multiple and extended selection modes
- single or multiple column display

HComb, the ComboBox Control -- includes all the features of the HEdit and HList controls, including use of the WinWidgets DataEngine and Hot-Linking feature. HComb has simple , drop-down and drop-down list styles.

HTool, the Toolbar/Palette Control -- makes building toolbars, status bars and floating palettes as easy as designing a dialog box. In fact, the procedure is the same. Simply design the toolbar in a dialog editor and call HToolCreate, specifying the name of the dialog template, the window, and the side of the window to which it will be attached. HTool relays control notifications to your application, becoming invisible to your program. It's that easy!

HGrid, the Grid/Database Table Control -- designed specifically to make displaying database tables for browsing and editing as quick and painless as possible. HGrid employs the other WinWidgets, utilizing their 3D appearance and data manipulation capabilities to create a control that is easy and familiar to users and programmers alike. To the user, it presents a flexible, attractive interface, featuring:

- resizeable rows and columns
- drag-and-drop positioning of columns
- non-scrolling columns
- ability to copy and paste to spreadsheet or word processor
- editing in-place or spreadsheet-style, in the toolbar

HGrid's programming interface simplifies the manipulation of data and the control of user input. HGrid supports:

- edit, list, combobox and checkbox fields
- child, MDI child and pop-up implementations
- browse-only mode that can be set for the entire grid or individual fields or records
- addition, deletion and insertion of entire records in a single statement

- full support for record buffering through a simple callback procedure

The following chapters describe each of the WinWidgets in detail, including their attributes, methods, events and coding examples. The [DataEngine](#) chapter describes the data types supported by the WinWidgets, along with their formatting options and formatting examples.

## How do I... ?

- use this Manual?
- install the WinWidgets?
- use the WinWidgets with Visual C++?
- use the MFC Class Wrappers?
- use a VBX control in MFC?
- use the WinWidgets with my Design Tools?
- control the WinWidgets at Run-time?
- connect the WinWidgets to my Data?
- respond to Events?
- use my own Resources?
- subclass the WinWidgets?
- get Technical Support?

## Controlling the WinWidgets at Run-time

Once a form or dialog box has been designed using the procedures described in [Integration, Layout and Design](#), there are just a few steps necessary to get up and running:

- 1) Initialize the WinWidgets library by calling the WidgetsInit() function. WidgetsInit() registers the WinWidgets class names with Windows, allowing the controls to be instantiated. This step is not required when using the VBX versions of the controls.
- 2) Initialize the run-time properties of individual controls when the form or dialog is loaded. The location of control-initialization code depends on the programming environment:

### C API

Initialize controls in response to the WM\_INITDIALOG message within a dialog procedure. At this point the controls have been created, but not displayed.

### MFC

Initialize controls by overriding the CDialog::OnInitDialog() member function. This function is called in response to the WM\_INITDIALOG message.

### OWL

Initialize controls by overriding the TDialog::WMInitDialog() member function. This function is called in response to the WM\_INITDIALOG message.

### Visual Basic

Initialize controls in the form's Load procedure, *Sub Form\_Load()*.

Initialization of a control often involves moving data to the control from application variables or a database. There are several approaches to this process, which we discuss in [Connecting the WinWidgets to Data](#).

- 3) Respond to control events to interact with the user dynamically. All of the controls except the Static and Toolbar controls can trigger events in response to user actions. For instance, when a user changes the selection in a Grid control, a SelChange event is triggered. The list of events for each control can be seen by pressing the button next to the word **Events** on the control's introductory page. General procedures for handling events in various programming environments are discussed in the [Handling Events](#) topic.
- 4) Retrieve information from the controls upon closing the form or dialog with an OK button or other positive response. This step can be avoided in C/C++ if the data is [Hot-Linked](#) to application variables. As with Step #2, there are several approaches to this process, which we discuss in [Connecting the WinWidgets to Data](#).

## Connecting the WinWidgets to Data

Each of the WinWidgets controls provides convenient methods for data-related tasks. Some of the more common tasks are listed below. Press the buttons to see the related topics:

- getting and setting the data from a CheckBox or RadioButton
- getting and setting the data from an Edit control
- adding, inserting, deleting and selecting items in a ListBox or ComboBox
  - When using the ListBox or ComboBox, remember that they both support non-displayed data associated with list items. See [Using Codes in the ListBox and ComboBox](#) for details.
- adding, inserting and deleting records in the Grid
- adding, inserting and deleting fields in the Grid
- implementing a record buffer in the Grid

In addition, the WinWidgets provide solutions that are tailored to the development environment wherever possible. For instance, in Visual Basic 3.0, the WinWidgets can be connected to a *data control* for automatic initialization and retrieval of user input. These tailored solutions are listed below:

### C API

The WinWidgets C language API allows the controls to be "Hot-Linked" to application variables, which are automatically updated in response to user input. See [Hot-Linking the WinWidgets](#) for more details.

### MFC

When using MFC, the best method for connecting the WinWidgets to your data depends on the control interface you have chosen -, either the WinWidgets as Visual Basic controls (VBX's) or as old- style custom controls. As VBX controls, the WinWidgets support the DDX/DDV services provided by MFC through the ClassWizard. For more information about using DDX/DDV with VBX controls, consult the MFC documentation and MFC Tech Note #26.

As non-VBX custom controls, the WinWidgets' MFC Class Wrappers encapsulate the data connectivity provided through the C language API. The MFC Class Wrappers also support [Hot-Linking](#) the WinWidgets to application variables.

### OWL

Our OWL Wrappers support the use of Transfer Buffers through data transfer routines similar to those used in other OWL control classes. The use of Transfer Buffers is covered in the Object Windows Users Guide. The sample application in the SSTRANS.EXE provided with WinWidgets is an example of WinWidgets used with a Transfer Buffer.

The OWL Class Wrappers also encapsulate the data connectivity provided through the C language API, including [Hot-Linking](#) the WinWidgets to application variables.

### Visual Basic

In Visual Basic version 3.0, the WinWidgets can be connected to the Visual Basic Data Control, which can, in turn, be connected to a variety of databases. The WinWidgets can be used to display values from a single field of a single database record (Edit and CheckBox), multiple records from a single field (ListBox and ComboBox), or entire database tables (the Grid). For more details about using the WinWidgets in Visual Basic version 3.0, see [WinWidgets and the Visual Basic Data Control](#).



## The WinWidgets DataEngine

The DataEngine is used by the WinWidgets to convert between illegible binary data and text. The methods used in these conversions depend on the data's classification, or the *data class*. Data are grouped into classes based on their meaning in the real world. For example, Dates, Times, Numbers, and Strings are data classes.

Within a data class, there are common formats used to present the data for viewing and editing. For Dates, these formats include "m/d/yy" and "Mmm d, yyyy." Typically, as in these examples, the format is defined as a character string.

There are also common ways to store data of a particular class. For instance, Dates can be stored as three integers representing the year, month and day, or as a long integer representing the number of days from an arbitrary starting date. These different binary representations of data are called *data types*.

The following sections describe the formats and data types for each data class:

---

### Contents



**Data Class  
Overview**



**Numbers and Currency**



**Booleans**



**Strings**



**Dates and Times**



**Color Indicators**



**Masked Strings**

## Data Classes

The WinWidgets use a single character code to identify each data class, and each data type within the class. When developing an application, each control's data class, type and format can be selected at design time. First, the data class is selected from the available list, then the acceptable data types and sample formats are displayed. The character codes for the data class and type and the format string are concatenated and stored as the window text of the control.

<b><u>Class ID</u></b>	<b>Indicator</b>	<b>Description</b>
<u>HC_BOOL</u>	b	Binary choices (true/false, male/female, on/off)
HC_CHAR	h	single character
<u>HC_CLOCK</u>	k	system date and/or system time, updated automatically
<u>HC_CURRENCY</u>	c	monetary value
<u>HC_DATE</u>	d	date (year, month, day)
<u>HC_DATETIME</u>	a	date and time (year, month, day, hour, minute, second, millisecond)
<u>HC_MASK</u>	m	character string with an edit mask
<u>HC_NUMBER</u>	n	numerical value
<u>HC_STRING</u>	s	NULL-terminated character string



HC\_TIME

t time (hour, minute, second, millisecond)

## Data Types

<u>Type ID</u>	<b>Indicat or</b>	<b>Description</b>
HT_CHAR	h	char (8 bit integer)
HT_BYTE	b	unsigned char
HT_SHORT	s	short int (16 bit integer)
HT_WORD	w	unsigned short int
HT_INT	i	int
HT_UINT	u	unsigned int
HT_LONG	l	long (32 bit integer)
HT_DWORD	W	unsigned long
HT_FLOAT	f	float (32 bit floating point)
HT_DOUBLE	d	double (64 bit floating point)
<u>HT_BIGMONEY</u>	g	64 bit signed integer, used by SQL, a value of 1 represents 1/10,000 of a currency unit.
HT_STRING	s	NULL-terminated character string
<u>HT_DATETIME</u>	D	used by the DataEngine
<u>HT_ODBCTIMESTAMP</u>	s	ODBC-defined date/time struct
<u>HT_SQLDATETIME4</u>	q	4 byte date/time fr SQL/Server
<u>HT_SQLDATETIME</u>	Q	8 byte date/time fr SQL/Server
<u>HT_TM</u>	m	date/time struct fr <time.h>
<u>HT_TIME_T</u>	T	32 bit date/time fr <time.h>
<u>HT_DOUBLEDATE</u>	u	64 bit date/time used by Excel
<u>HT_ODBCDATE</u>	d	ODBC-defined date struct
<u>HT_DOSDATE</u>	a	DOS system date struct
<u>HT_DOSFILEDATE</u>	A	DOS file date struct
<u>HT_LONGDATE</u>	l	32 bit date value
<u>HT_ODBCTIME</u>	t	ODBC-defined time struct
<u>HT_DOSTIME</u>	i	DOS system time struct
<u>HT_DOSFILETIME</u>	l	DOS file time struct

## Color Formatting

One formatting option that is common across all data classes is the inclusion of text color indicators. Color indicators are enclosed in braces [], and should appear at the beginning of the format string. The sixteen recognized color names are listed below. Case is not important. A color indicator may also be an RGB triple such as [0, 255, 255], in which the red, green and blue components of the color are given as integers between 0 and 255.

<b>Indicator</b>	<b>Color</b>	<b>Indicator</b>	<b>Color</b>
black	black	gray	dark gray
white	white	ltgray	light gray
red	red	dkred	dark red
green	green	dkgreen	dark green
blue	blue	dkblue	dark blue
yellow	yellow	olive	olive
magenta	magenta	purple	purple
cyan	cyan	drab	dark cyan

## The Boolean Data Class

Boolean type formatting provides an informative and attractive interface to a simple yes or no choice. The Format string contains substrings for the TRUE and FALSE conditions, separated by a semicolon. The keyboard interface allows the user to select true by typing the first letter of the TRUE substring, select false with the first letter of the FALSE substring, or toggle between states with the space bar.

The data types supported by the HC\_BOOL class are listed below. A zero data value is interpreted as false, any other is true.

### Data Types

<b>Type ID</b>	<b>Indicator or</b>	<b>Description</b>
HT_CHAR	h	char (8 bit integer)
HT_BYTE	b	unsigned char
HT_SHORT	s	short int (16 bit integer)
HT_WORD	w	unsigned short int
HT_INT	i	int
HT_UINT	u	unsigned int
HT_LONG	l	long (32 bit integer)
HT_DWORD	W	unsigned long

### Examples

<b>Format</b>	<b>Data</b>	<b>Formatted Text</b>
True;False	1	True
Male;Female	0	Female
Yes:[red]No!	0	No!

## The Number and Currency Data Classes

For numerical and monetary types, the DataEngine uses token replacement to create the formatted Text. Tokens are symbolic placeholders that are replaced by digits or other characters when the number is formatted as text. The tokens and their replacements are

described below.

The thousands separator and decimal indicator are obtained from the Win.ini file and can be edited using the Windows Control Panel.

Any non-token characters in the format string are copied literally into the Text. If no format string is specified, the string "\*0.\*" is used.

## Data Types

Type ID	Indicat or	Description
HT_CHAR	h	char (8 bit integer)
HT_BYTE	b	unsigned char
HT_SHORT	s	short int (16 bit integer)
HT_WORD	w	unsigned short int
HT_INT	i	int
HT_UINT	u	unsigned int
HT_LONG	l	long (32 bit integer)
HT_DWORD	W	unsigned long
HT_FLOAT	f	float (32 bit floating point)
HT_DOUBLE	d	double (64 bit floating point)
<u>HT_BIGMONEY</u>	g	64 bit signed integer, used by SQL, a value of 1 represents 1/10,000 of a currency unit.

## Tokens

Code	Description
, (comma)	Inserts the International Thousands Separator every three places to the left of the decimal.
. (period)	Is replaced with the current International Decimal Indicator.
; (semi-colon)	Separates formats for positive and negative numbers.
#	Is replaced by no digits or one digit.
*	Is replaced by zero or more digits.
0 (zero)	Is replaced by one digit or a "0" (zero).
\	Causes the next character to be treated as a literal.
[]	Used to enclose a color indicator.

## Examples

Data	Format	Formatted Text
1001.536	\$\$,##0;[RED](\$\$,##0)	\$1,002
-1001.536	\$\$,##0;[BLUE](\$\$,##0)	(\$1,002)
-0.2	0.00	-0.20
123	00000	00123
104	->*<-	->104<-

30.2500	*0.*	30.25
12	+*0.00;-*0.00	+12.00

## The Date, Time, DateTime and Clock Data Classes

For date, time and date/time types, the DataEngine uses token replacement to create the formatted Text. The tokens and their replacements are described below. In Edit mode, dates and times are displayed using the International ShortDate and Time formats listed in the Win.ini file and edited through the Windows Control Panel. In Display mode, the Format string is parsed for tokens. Any non-token characters are copied literally into the Text. Some of the tokens are case sensitive, such as mmm, and others are affected by the Windows international settings, such as "/" and ":". If no Format string is specified, the International ShortDate and Time formats are used in Display mode as well. The acceptable data types for the date/time classes are:

### Data Types

Type ID	Indicat or	Description
<u>HT_DATETIME</u>	D	used by the DataEngine
<u>HT_ODBCTIMESTAMP</u>	s	ODBC-defined date/time struct
<u>HT_SQLDATETIME4</u>	q	4 byte date/time fr SQL/Server
<u>HT_SQLDATETIME</u>	Q	8 byte date/time fr SQL/Server
<u>HT_TM</u>	m	date/time struct fr <time.h>
<u>HT_TIME_T</u>	T	32 bit date/time fr <time.h>
<u>HT_DOUBLEDATE</u>	u	64 bit date/time used by Excel
<u>HT_ODBCDATE</u>	d	ODBC-defined date struct
<u>HT_DOSDATE</u>	a	DOS system date struct
<u>HT_DOSFILEDATE</u>	A	DOS file date struct
<u>HT_LONGDATE</u>	l	32 bit date value
<u>HT_ODBCTIME</u>	t	ODBC-defined time struct
<u>HT_DOSTIME</u>	i	DOS system time struct
<u>HT_DOSFILETIME</u>	l	DOS file time struct

### Tokens

Token	Sample	Description
m	12	month (1 - 12)
mm	07	month (01 - 12)
mmm	Feb	month (Jan - Dec)
mmmm	February	month (January - December)
d	31	day (1 - 31)
dd	03	day (01 - 31)
ddd	Fri	day-of-week (Sun - Sat)
dddd	Friday	day-of-week (Sunday - Saturday)
yy	91	year (00 - 99)

yyyy	1991	year (1700 - 2900)
/	/	International Date Separator
h	10	hour (0-23) or (1-12) if AP or ap is used
hh	01	hour (00-23) or (1-12) if AP or ap is used
mm	02	minute (00-59), only after h or hh
ss	01	seconds (00-59)
fff	001	milliseconds (000-999)
ap	am	am or pm (or international equivalent)
:(colon)	:	International Time Separator
\		The next character is treated as a literal.

## Examples

Data	Format	Formatted Text
9/22/92 14:05	Ddd - Mmm d, yyyy	Tue - Sep 22, 1992
9/22/92 14:05	m/d/yy h:mm AP	9/22/92 2:05 PM
9/22/92 14:05	Mmm d - h:mm ap	Sep 9 - 2:05 pm

## The Mask Data Class

Mask formatting provides a powerful way to control user input with character strings. The format string may be a combination of literal characters, which appear as themselves, and tokens, which appear as blanks (or zeros for digits) and may be replaced by appropriate user-input. The tokens and their permissible replacements are listed below. The logic for creating the Text string from the Data string is as follows: For each format character, if the character is a token, copy the next acceptable Data character to the Text string. If there are no more acceptable Data characters, use a blank or zero. If the format character is not a token copy it to the Text string, and if the Data character matches the format character move to the next Data character.

## Data Types

Type ID	Indicator	Description
HT_STRING	s	NULL-terminated character string

## Tokens

Token	Permissible Replacements
#	Any numeric digit (0-9).
@	Any alphabetic character (a-z, A-Z)
!	Any punctuation character
*	Any single printable character.
\	Causes the next character to be treated as a literal.

## Examples

Data	Format	Formatted Text
"012345678"	###-##-####	012-34-5678
"012-34-5678"	###-##-####	012-34-5678
"2125551212"	(###) ###-####	(212) 555-1212
"212-555-1212"	(###) ###*####	(212) 555-1212
"Feb-1992"	@@@, ####	Feb, 1992

## The String Data Class

The string class uses the format string to set a maximum text length for edit mode and/or to initialize the control's Text. After initialization, the format string serves no purpose.

### Data Types

Type ID	Indicator	Description
HT_STRING	s	NULL-terminated character string

### Examples

Format	Data	Formatted Text	Text Limit
[magenta]Text	none	Text	none
[magenta]Text	New text.	New text.	none
%5Text	none	Text	5

```
typedef struct tagDATETIME
{
short   year;    // (i.e. 1970)
BYTE    month;  // 1-12
BYTE    day;    // 1-31
BYTE    hour;   // 0-23
BYTE    minute; // 0-59
BYTE    second; // 0-59
WORD    msec;   // 0-999
}
DATETIME,
far *LPDATETIME;
```

```
typedef struct tagODBCDATE
{
    short   year;
    WORD    month;
    WORD    day;
}
ODBCDATE,
far *LPODBCDATE;
```

```
typedef struct tagODBCTIME
{
    WORD    hour;
    WORD    minute;
    WORD    second;
}
ODBCTIME,
far *LPODBCTIME;
```



```
typedef struct tagODBCTIMESTAMP
{
    short    year;
    WORD    month;
    WORD    day;
    WORD    hour;
    WORD    minute;
    WORD    second;
    DWORD   fraction;
}
ODBCTIMESTAMP,
far *LPODBCTIMESTAMP;
```

```
typedef struct tagSQLDATETIME4
{
    WORD    days; //Julian days from 1/1/1900 to 6/6/2079
    WORD    time; //seconds since midnight
}
SQLDATETIME4,
far *LPSQLDATETIME4;
```

```
typedef struct tagSQLDATETIME
{
    long    days; //Jul days since 1/1/1900 from
              1/1/1753-12/31/9999
    long    time; //milliseconds since midnight
}
SQLDATETIME,
far *LPSQLDATETIME;
```

```
typedef long TIME_T, // seconds since Jan 1, 1970
far *LPTIME_T; // at 00:00:00
```

```
typedef struct tagTM
{
    short  tm_sec;    //seconds after the minute (0-61?)
    short  tm_min;    //minutes after the hour (0-59)
    short  tm_hour;   //hours since midnight (0-23)
    short  tm_mday;   //day of the month (1-31)
    short  tm_mon;    //months since January (0-11)
    short  tm_year;   //years since 1900
    short  tm_wday;   //days since Sunday (0-6)
    short  tm_yday;   //days since January 1 (0-365)
    short  tm_isdst;  //Daylight Saving Time, >0 if DST
}
TM,
far * LPTM;
```

```
typedef struct tagDOSDATE
{
    BYTE    day;        // 1-31
    BYTE    month;     // 1-12
    WORD    year;       // 1980-2099
    BYTE    dayofweek; // 0-6, 0=Sunday
}
DOSDATE,
far * LPDOSDATE;
```

```
typedef struct tagDOSTIME
{
    BYTE    hour;        // 0-23
    BYTE    minute;     // 0-59
    BYTE    second;     // 0-59
    BYTE    hsecond;    // 0-99
}
DOSTIME,
far * LPDOSTIME;
```

```
typedef struct tagDOSFILEDATE
{
WORD    Year   : 7; // years since 1980
WORD    Month  : 4; // month (1-12)
WORD    Day    : 5; // day (1-31)
}
DOSFILEDATE,
far * LPDOSFILEDATE;
```



```
typedef struct tagDOSFILETIME
{
    WORD    Hour   : 5; // hour (0-23)
    WORD    Min    : 6; // minutes (0-59)
    WORD    Sec    : 5; // secs/2 (10 here means 20)
}
DOSFILETIME,
far * LPDOSFILETIME;
```

```
typedef double DOUBLEDATE, // real number of days since  
                far * LPDOUBLEDATE; // midnt, December 30, 1899*
```

```
// Excel, and perhaps other spreadsheets, claim that  
// their date represents the number of days from  
// December 31, 1899 - making January 1, 1900  
// day 1. However, they neglect the fact that 1900 was  
// NOT a leap year! There was no 2/29/1900, which is a  
// valid date in Excel. In addition, SQL uses the  
// number of days SINCE January 1, 1900 - making  
// 1/1/1900 day 0. The net effect is that the integral  
// portion of a DOUBLEDATE will be 2 greater than the  
// days element of the SQL types.
```

```
typedef long LONGDATE, // year*10000 + month*100 + day
far *LPLONGDATE; // e.g. 12/30/1968 -> 19681230
```

```
typedef struct
{
    DWORD  bm[2];  // bm[0] is the more significant DWORD
}
BIGMONEY,
far * LPBIGMONEY;
```

```
typedef struct
{
    WORD bwm[4];    // bwm[3] is the most significant WORD
}
BMWWORDS,
far *LPBMWWORDS;
```

The HC\_CLOCK data class sets a one-second timer if the format string contains a time, otherwise it sets a one-minute timer. With each timer message the control updates its data based on the system time and redisplay the text. The HC\_CLOCK class makes it extremely easy to incorporate a clock or calendar in an application's status bar or ribbon.

## **Case Sensitive Tokens**

These tokens are case sensitive. The DataEngine interprets the token's capitalization as all capitals, all lowercase, or first letter capitalized.

## Frequently Asked Questions

### General

- Where is my manual?!
- Why won't my dialog with WinWidgets controls come up?
- Do the WinWidgets respond to the same messages as the built-in Windows controls?
- Is there any way to avoid redistributing WIDGETS.DLL with my application?
- Why do static controls paint over other controls in a dialog?
- Bitmaps and Icons appear in buttons and static controls in my design environment but are missing at run-time. Why?
- Why is there no Toolbar control button in the Resource Workshop/Dialog Editor/AppStudio?
- How do I get rid of the "Control Style Expected" warning message in Borland's Resource Workshop?
- How do I prevent infinite loops when I respond to events?

### Visual Basic

- Why is there no Data property in the VBX WinWidgets?

### C++

- When I link my MFC application I get a bunch of the following error message: "error L2044: .... : symbol multiply defined, use /NOE". What is going on?
- When I link my MFC application I get the following error message for every function in the library: "error L2029: ... : unresolved external". What is going on?

### Edit Control

- How do I limit the number of characters in an edit control?
- Why don't Insert and Delete work in masked edit fields?

### Grid Control

- I add/insert/delete records in the Grid and nothing happens. What is going on?
- Why does my grid come up in my application without any fields?
- Why does my grid come up in my application without any records?
- The Grid appears with data in the wrong column or garbage in some columns. Why?
- Is it possible to get rid of the scroll bars in the Grid?
- How do I get rid of the white space on the right-hand side of the Grid?



## Where Is My Manual?

To print all or part of this document from electronic form, choose **File, Print Manual...**. Select from the list of chapters and chapter sections those you want to have on paper. Then press **Print**. You may cancel the process at any time by pressing **Cancel**.

Without the trials of producing a printed manual, we've had time to make vast improvements to the WinWidgets' documentation and the tools themselves. The current release offers many new features, more explanations and more examples. Our new on-line manual is designed for easy use as both a quick-reference and a general guide. Unlike a printed document, the on-line manual can be easily updated and expanded, it contains extensive cross-referencing and search capabilities, and it doesn't take any space on your desk.

If you've never used an on-line manual in the Windows Help System, we're glad to finally introduce you. Among other features, Windows Help provides:

- hyperlinks between related topics
- alphabetic topic searches (use the **Search** button to type in keywords, then double-click on a word or press the **Show Topics** button, select a topic and press **Go To**.)
- colored text and pictures
- pop-up definitions of important words and constants
- bookmarking and annotation that won't wear out the pages (check out the **Bookmark** and **Edit** menus)
- copy and paste example text from the manual directly to your programs

If that's not enough, Windows Help allows us to integrate the manual with your design tools so that when you design screens with the WinWidgets, context-sensitive help is only an F1 click away. For more information on using these features, see [Using the Manual](#).

## **Why won't my dialog with WinWidgets controls even come up?**

This is usually caused because the WinWidgets window classes have not been properly registered. Make sure you are calling `WidgetsInit()` at the beginning of your application.

## **Do the WinWidgets respond to the same messages as the built-in Windows controls?**

WinWidgets controls are not subclassed from standard Windows controls; we wrote them all from scratch. Therefore, they do not default to the standard Windows controls processing for the standard control messages (i.e. BM\_XXXX, EM\_XXXX, LBM\_XXXX). WinWidgets also do quite a bit more than the standard Windows controls and require a larger message interface. They do respond to certain Windows messages, such as WM\_SETTEXT, WM\_CUT, WM\_COPY and WM\_UNDO. These cases are documented in this manual.

## **Is there any way to avoid redistributing WIDGETS.DLL with my application?**

Yes. If you have purchased the WinWidgets source code, we will provide a makefile for compiling a static version of the library, which you can link directly into your .EXE file. The advantages to static linking are that you don't link the entire content of WIDGETS.DLL into your app, only the code that you actually use. Another advantage is that your application will be immune to the effects of other applications installing different versions of WIDGETS.DLL in a common directory.

## **Why do static controls paint over other controls in a dialog?**

Windows paints controls in a dialog box according to the order in which they appear in the dialog resource. This order can be set in most design environments, so you don't have to be that careful about the order in which you include the controls initially. Just make sure that when you save the resource, any static controls appear after any other controls that should appear over them.

## **Bitmaps and icons appear in buttons and static controls in my design environment but are missing at run-time.**

### **Why?**

Make sure that your bitmaps and icons are compiled as resources in your application or reside in the current working directory. The working directory may be different at run-time than at design-time, which is why bitmaps can appear properly in one instance and not another. The resource editors do not load a copy of your application and thus can only get bitmaps and icons from files in the working directory.

## **Why is there no Toolbar control button in the Resource Workshop/Dialog Editor/AppStudio?**

The Toolbar is not strictly a control, rather it is a modeless dialog containing other controls. Create a toolbar by designing the dialog in your favorite resource editor and passing the dialog resource to the Toolbar's Create method.

## How do I get rid of the "Control Style Expected" warning message in Borland's Resource Workshop?

The Resource Workshop is looking for the numerical values of WinWidgets' constants, which are located in the WinWidgets header, WIDGETS.H. To prevent the message, add WIDGETS.H to your project by choosing **File, Add to Project...** You may also need to modify the Resource Workshop's include path under **File, Preferences** when all projects are closed.



## **How do I prevent infinite loops when I respond to events?**

Occasionally, responding to an event can produce the same event, which initiates an infinite loop. To avoid the recursion tell the control to be quiet (suppress events) while you are responding to the event. All of the WinWidgets have a Quiet attribute that can be set to TRUE at the beginning of your procedure and FALSE at the end.

## **Why is there no Data property in the VBX WinWidgets?**

Unlike C and C++, Visual Basic is not a strongly-typed language. Visual Basic programmers can assign text strings to binary data types and rely on the language to do the conversion for them. Furthermore, Visual Basic does not support most of the data types supported by WinWidgets. For these reasons, our VBX interface does not provide access to the Data attribute of the controls. We do, however, provide access to the DataType attribute through the VBX interface so that C++ programmers designing in AppStudio can have control over the data types their controls use. The C++ classes we wrote do not rely on the VBX interface and thus do provide full access to the controls' Data.

If all of this makes you wonder why Microsoft thinks C++ programmers will feel more comfortable programming with VBX controls, you're not the only one.

**When I link my MFC application I get a bunch of the following error message: "error L2044: .... : symbol multiply defined, use /NOE". What is going on?**

This is probably caused by compiling a debugging version of your app with the version of the Wrapper libraries supplied on our distribution disks, which are release versions. You must recompile these libraries for debugging with the makefiles we supply.

**When I link my MFC application I get the following error message for every function in the library: "error L2029: ... : unresolved external". What is going on?**

This is probably caused by a mismatch between the memory model of your application and the Wrapper library. MFCWDGSL.LIB is a large model library; MFCWDGSM.LIB is a medium model library.

## How do I limit the number of characters in an edit control?

The best way to set a maximum text length for a string in an edit control is to set the MaxTextLen attribute for the control. Some users have attempted to limit the number of characters by employing a masked edit with a mask of the maximum length. We do not recommend this approach because masked edit controls behave in ways that are generally undesirable for generic strings. Masks are better used for enforcing some unusual string syntax, not for simply setting a maximum length.

## **Why don't Insert and Delete work in masked edit fields?**

Masked edit fields are typically used to enforce a strict string syntax that is inconvenient for the user to remember each time he or she types, such as the pattern of hyphens and parentheses in a phone number. For this reason, highlighting text and deleting in a masked edit field actually replaces non-masked positions with whitespace and leaves masked positions untouched. Similarly inserting one character into a mask where three characters have been highlighted will replace the first of the three characters with the inserted character and the remaining two with whitespace. This behavior is appropriate because it enables the user to see what parts of the string have been deleted or altered, while still preserving the string syntax. This is why using a mask to simply limit string length is a bad idea.

## **I add/insert/delete records in the Grid and nothing happens. What is going on?**

This problem is usually caused by failing update the Grid. Changes in the Grid are not reflected in the display until the Grid is updated.

## **Why does my grid come up in my application without any fields?**

This usually means the WinWidgets cannot find the Grid resource in your application or .GRS file in the working directory. If you have compiled a GRS file as a resource and this problem persists, make sure that you have not used a "#define" directive for the resource name. It should be used as a string, not as an integer.



## **Why does my grid come up in my application without any records?**

The Dialog Editor, Resource Workshop, and AppStudio add records to the Grid in their design environments for the purpose of testing. These records are not added to the Grid by default at run-time. You must add records explicitly.

## **Why is data in the Grid appearing in the wrong column or otherwise appearing corrupted?**

Applications using the Grid must be compiled with Single Byte Alignment. This problem typically occurs when Double Byte Alignment is set and one of the fields is defined with an odd byte size. See [HGrid Record Structures](#) for more details.

## **Is it possible to get rid of the scroll bars in the Grid?**

Yes. The Grid responds to the standard Windows `WS_HSCROLL` and `WS_VSCROLL` styles, which you can set at design time. Grids designed using previous versions WinWidgets will have these styles set by default, but can be resaved as version 2.0 grids with these styles turned off.

## How do I get rid of the white space on the right-hand side of the Grid?

To have a Grid completely fill its window without showing empty space, make sure the fields are sized appropriately so that they combine to fill the window's client area. If a vertical scroll bar is enabled, leave room for the scroll bar at the right and make sure the DisableNoScroll style is set to prevent the scroll bar from disappearing when it is not needed.

## **HButt, The WinWidgets Button Control**

- Attributes
- Events

HButt is an-all-purpose button control. Use it as a pushbutton, radiobutton, checkbox, or multiple-state button. It displays multiple lines of text and/or bitmaps and icons. It supports mnemonics and the default pushbutton style. It even plays sound resources. As a checkbox or radiobutton, HButt can be Hot-Linked to a data source that is automatically updated whenever the button changes state. All of the standard button types have a 3D appearance by default, but HButt can be easily tailored with custom images, styles and formatting.

### **Additional Topics**

- Using custom pictures and sounds
- Hot-Linking a button to your data
- Using HButt with the Visual Basic Data Control
- Displaying 256-color bitmaps

## HButt Attributes

<u>AutoAdvance</u>	<u>MaskColor</u>	<u>SoundMode</u>
<u>Background</u>	<u>NoButton</u>	<u>Squared</u>
<u>BtnType</u>	<u>NoFocus</u>	<u>State</u>
<u>Count</u>	<u>Palette</u>	<u>Text</u>
<u>Data</u>	<u>PicAlign</u>	<u>TextAlign</u>
<u>DataLink</u>	<u>Picture</u>	<u>TextColor</u>
<u>DataType</u>	<u>Pressed</u>	<u>TextIndent</u>
<u>DownPics</u>	<u>Quiet</u>	<u>TextJustify</u>
<u>Font</u>	<u>RelAlign</u>	<u>Transparency</u>
<u>LastInGroup</u>	<u>Sound</u>	

## AutoAdvance Attribute

When set, the button advances one State in its state cycle each time it is pressed.

### Usage

#### C/C++

Window Style: HBS\_AUTOADVANCE

#### VBX

[*form.*][*control.*]**AutoAdvance**

### Remarks

The AutoAdvance attribute does not apply to PushButtons. This attribute is read-only at run time.

## Background Attribute

The color or pattern used to paint the background of checkboxes and radio buttons and the corners of pushbuttons

### Usage

#### C

```
hbrBkgnd = (HBRUSH)SendMessage(hWnd, HBM_GETBKGNDBRUSH, 0, 0L);  
hbrOldBkgnd = (HBRUSH)SendMessage(hWnd, HBM_SETBKGNDBRUSH,  
(WPARAM)hNewBrush, 0L);
```

#### C++

##### OWL

```
hbrBkgnd = [THButtonObj.]GetBkgndBrush(void);  
hbrOldBkgnd = [THButtonObj.]SetBkgndBrush(hbrNewBrush);
```

##### MFC

```
pBkgnd = [CHButtonObj.]GetBkgndBrush(void);  
pOldBkgnd = [CHButtonObj.]SetBkgndBrush(pNewBrush);
```

#### VBX

[*form.*][*control.*]**BackColor**[ = *color* ]

See *Visual Basic Language Reference*, "BackColor, ForeColor Properties"

### Arguments/Parameters

HBRUSH hbrNewBrush	Handle of a new brush to be set as the background brush
CBrush pNewBrush	Pointer to a CBrush object containing the new background brush handle

### Return values

HBRUSH hbrBkgnd	Handle to the current background brush
HBRUSH hbrOldBkgnd	Handle to the previous background brush
CBrush *pBkgnd	Pointer to a CBrush object containing the current background brush handle
CBrush *pOldBkgnd	Pointer to a CBrush object containing the previous background brush handle

### Remarks

C and C++ applications are responsible for destroying any brushes they create. HButton also supports WM\_CTLCOLOR processing; see the Windows SDK documentation for details

## BtnType Attribute

Determines the type of button (i.e pushbutton, radio, checkbox, defpushbutton, etc.)

### Usage

#### C/C++

Window Styles:

HBS\_PUSHBUTTON

HBS\_DEFPUSHBUTTON

HBS\_CHECKBOX

HBS\_RADIOBUTTON

HBS\_3STATE

HBS\_GROUPPUSH

#### VBX

Not Used. The various button types are implemented as separate controls in Visual Basic.

### Remarks

The BtnType attribute is read-only at run time.



## Count Attribute

The total number of Pictures (including the "pressed" pictures if the DownPics attribute is on) displayed by the button in a complete cycle

### Usage

#### C

```
iCount = (int)SendMessage(hWnd, HBM_GETCOUNT, 0, 0L);
```

#### C++

```
iCount = [CHButton.]GetCount();
```

#### VBX

Not Used. The number of pictures is determined by the type of button and the setting of the DownPics attribute.

### Return values

`int iCount`

The number of pictures stored by the button

### Remarks

The Count attribute is read-only.

## Data Attribute

The native (binary) data displayed by a checkbox, of a type support by the [boolean data class](#) as defined in the [DataEngine](#) chapter. [States](#) are numbered starting at zero for unchecked, one for checked, two for greyed (in a 3-State button), etc. For two state buttons, any non-zero data values correspond to a checked state.

### Usage

#### C

```
IBytesCopied = SendMessage(hWnd, HBM_GETDATA, (WPARAM)iMaxBytes,  
(LPARAM)lpData);
```

```
IBytesCopied = SendMessage(hWnd, HBM_SETDATA, 0, (LPARAM)lpData);
```

```
iSize = (int)SendMessage(hWnd, HBM_GETDATASIZE, 0, 0L);
```

#### C++

```
IBytesCopied = [CHBCheckObj/CHRadioObj].GetData(lpData [, iMaxBytes = 0 ] );
```

```
IBytesCopied = [CHBCheckObj/CHRadioObj].SetData(lpData);
```

```
iSize = [CHBCheckObj/CHRadioObj].GetDataSize();
```

#### VBX

Not Used. In Visual Basic 3.0 the checkbox and radiobutton can be connected directly to a DataSource and DataField. See [Data-Awareness](#).

### Arguments/Parameters

void FAR \*lpData

Pointer to data

int iMaxBytes

Maximum number of bytes to copy (used only for strings)

### Return values

LONG lBytesCopied

Number of bytes actually copied to or from the control

int iSize

The size of the Data

### See Also

[DataLink](#), [State](#)

## DataLink Attribute

A pointer to the variable or buffer that is updated automatically when the user changes the Data attribute of a checkbox, or the State of a radiobutton

### Usage

#### C

```
lpLink = (void FAR *)SendMessage(hWnd, HBM_GETDATALINK, 0, 0L);  
lBytesCopied = (LONG)SendMessage(hWnd, HBM_SETDATALINK, bUseIndex,  
(LPARAM) (LPVOID) lpBuf);
```

#### C++

```
lpLink = [CHBCheckObj/CHBRadioObj/CHB3StateObj.]GetDataLink();  
lBytesCopied =  
[CHBCheckObj/CHBRadioObj/CHB3StateObj.]SetDataLink( [bUseIndex=FALSE][,  
lpBuf=NULL] );
```

#### VBX

Not Used. In Visual Basic 3.0 the checkbox and radiobutton can be connected directly to a DataSource and DataField. See [Hot-Linking](#)

### Arguments/Parameters

<code>void FAR *lpBuf</code>	Pointer to application data buffer that is to be updated with a checkbox's Data or a radiobutton's ID or index in a group.
<code>BOOL bUseIndex</code>	For radiobuttons, this flag determines the DataLink is updated with the control ID (FALSE), or the control index in the radiobutton group (TRUE).

### Return values

<code>void FAR *lpLink</code>	Pointer to the current data link. NULL if no DataLink has been set.
<code>LONG lBytesCopied</code>	Contains the number of bytes copied from lpBuf

### Remarks

For checkboxes, the button's Data will be set to the contents of lpBuf, which should contain a zero value for unchecked or a non-zero value for checked. For radiobuttons, the DataLink attribute need only be set for the first button in a group. HButt will check the radiobutton in the group corresponding to the ID or group index passed in lpBuf, depending on the value of bUseIndex.

### See Also

[Data](#), [State](#), [Hot-Linking](#), [Data-Awareness](#)



## DownPics

When set, there are two Pictures for each State, one for when the button is Pressed and one for unpressed.

### Usage

#### C/C++

Window Style: **HBS\_DOWNPICS**

#### VBX

[*form.*][*control.*]**DownPics**

### See Also

Pressed, State, Picture, Count

## Font Attribute

The font used by the control

### Usage

#### C

```
hfFont = (HFONT)SendMessage(hWnd, HBM_GETFONT, 0, 0L);  
hfOldFont = SendMessage(hWnd, HBM_SETFONT, (WPARAM)hfNewFont,  
(LPARAM)bRedraw);
```

#### C++

##### OWL

```
hfFont = [THButtonObj.]GetFont();  
hfOldFont[THButtonObj.]SetFont(hfNewFont [, bRedraw = TRUE] );
```

##### MFC

```
pFont = [CHButtonObj.]GetFont();  
pOldFont[CHButtonObj.]SetFont(pNewFont [, bRedraw = TRUE] );
```

#### VBX

```
[form.][control.]FontBold[= boolean]  
[form.][control.]FontItalic[= boolean]  
[form.][control.]FontName[= font]  
[form.][control.]FontSize[= points]  
[form.][control.]FontStrikethru[= boolean]  
[form.][control.]FontUnderline[= boolean]
```

See Visual Basic Language Reference, "FontName Property"

### Arguments/Parameters

HFONT hfNewFont	Handle of the font to be set
CFont *pFont	Pointer to a CFont object containing the handle to the font to be set
BOOL bRedraw	A value of TRUE causes the control to repaint immediately

### Return values

HFONT hfFont	Handle to the control's current font
HFONT hfOldFont	Handle to the control's previous font
CFont *hfFont	Pointer to a CFont object containing the handle to the control's current font
CFont *fOldFont	Pointer to a CFont object containing the handle to the control's previous font

### Remarks

C and C++ applications are responsible for destroying any fonts they create.

## LastInGroup Attribute

Determines if a radio button is the last in a group delineated by the WS\_GROUP style in a dialog template

### Usage

#### C

```
bResult = (BOOL)SendMessage(hWnd, HBM_ISLASTINGROUP, 0, 0L);
```

#### C++

```
bResult = [CHBRadioObj.]IsLastInGroup()
```

#### VBX

Not Used

### Return values

BOOL bResult

TRUE if the button precedes another control with WS\_GROUP

## MaskColor Attribute

A color value used to indicate transparency in bitmaps

### Usage

#### C

```
crMask = SendMessage(hWnd, HBM_GETMASKCOLOR, 0, 0L);  
SendMessage(hWnd, HBM_SETMASKCOLOR, 0, (LPARAM) crNewMask);
```

#### C++

```
crMask = [CHButton.]GetMaskColor();  
[CHButton.]SetMaskColor(crNewMask);
```

#### VBX

```
[form.][control.]MaskColor[= color]
```

### Arguments/Parameters

COLORREF crNewMask	The new MaskColor value
--------------------	-------------------------

### Return values

COLORREF crMask	The current MaskColor value
-----------------	-----------------------------

### Remarks

The MaskColor attribute is set by default to be light green, RGB (0, 255, 0). Masking causes a background color to show through portions of a bitmap, allowing non-rectangular bitmap images in the same manner as Windows icons.



## NoButton Attribute

When the NoButton attribute is set, the button is not drawn as a pushbutton, leaving the background as the system window color.

### Usage

#### C/C++

Window Style: **HBS\_NOBUTTON**

#### VBX

Not Used

## NoFocus Attribute

When set, the button will not receive the input focus upon being pressed.

### Usage

#### C/C++

Window Style: HBS\_NOFOCUS

#### VBX

[*form.*][*control.*]**NoFocus**

### Remarks

This attribute is read-only at run time.

## Palette Attribute

A handle to a 256-color palette for a DIB-type Picture

### Usage

#### C

```
hpPalette = SendMessage(hWnd, HBM_GETPALETTE, 0, 0L);
```

#### C++

##### OWL

```
hpPalette = [CHButtonObj.]GetPalette();
```

##### MFC

```
pPalette = [CHButtonObj.]GetPalette();
```

#### VBX

*VBX buttons are "palette aware"; they automatically realize their own palettes.*

### Return values

HPALETTE hpPalette

A handle to a logical palette for the control

CPalette \*pPalette

A pointer to a CPalette object containing the handle to control's logical palette

### Remarks

Typically, an application that displays 256-color images will select and realize the control's palette in response to WM\_PALETTECHANGED and WM\_QUERYNEWPALETTE messages. For additional information see the example below and Microsoft's Palette Self-Study Module, available through the Microsoft Developer Network.

### Examples



## Palette Attribute Example (C API)

```
case WM_PALETTECHANGED:
    if (hWndThis == (HWND) wParam)
        break;
    else //Another palette is being used
        // fall through to WM_QUERYNEWPALETTE

case WM_QUERYNEWPALETTE:
{
    HDC          hDC;
    HPALETTE     hOldPal, hControlPalette;
    int          nChanged;

    hDC          = GetDC (hWndChild);

    //Get the palette from the button control
    hControlPalette = (HPALETTE)SendMessage(hWndThis,
                                             101,
                                             HBM_GETPALETTE,
                                             0,
                                             0);

    //Select the control's palette into the device context
    hOldPal      = SelectPalette (hDC, hControlPalette,
                                 (msg == WM_QUERYNEWPALETTE) ?
                                 FALSE : TRUE);

    //Realize the palette
    nChanged = RealizePalette (hDC);
    //Select the old palette
    SelectPalette (hDC, hOldPal, TRUE);
    ReleaseDC     (hWndChild, hDC);

    //Repaint if necessary
    if (nChanged)
        InvalidateRect (hWndChild, NULL, TRUE);

    return nChanged;
}
```

## PicAlign Attribute

An alignment code representing the position of the Picture within the button's client area

### Usage

#### C

```
iPAlign = (int)SendMessage(hWnd, HBM_GETPALIGN, 0, 0L);  
SendMessage(hWnd, HBM_SETPALIGN, (WPARAM)iNewRAlign, 0L);
```

#### C++

```
iPAlign = [CHButtObj.]GetPALign();  
[CHButtObj.]SetPALign(iNewPALign);
```

#### VBX

```
[form.][control.]AlignPicture[= integer]
```

### Arguments/Parameters

<code>int iNewPALign</code>	The new alignment code
-----------------------------	------------------------

### Return values

<code>int iPAlign</code>	The current alignment code
--------------------------	----------------------------

### Remarks

#### Alignment Codes

At design time, the PicAlign attribute can be set from within the Dialog Editor, Resource Workshop, or Visual Basic, or can be specified in the WindowText directly.

## Picture Attribute

The bitmap(s) or icon(s) displayed by the button

### Usage

#### C

```
hPic = (HANDLE)SendMessage(hWnd, HBM_GETPIC, (WPARAM)iIndex, 0L);  
hOldPic = (HANDLE)SendMessage(hWnd, HBM_SETPIC, (WPARAM)iIndex,  
(LPARAM)MAKELONG(hNewPic, wType));
```

#### C++

```
hPic = [CHButton.]GetPic([iIndex = 0]);  
hOldPic = [CHButton.]SetPic(hNewPic, wType [, iIndex = 0]);
```

#### VBX

```
[pushbutton.]Pic[= picture]  
[pushbutton.]Pic_Pressed[= picture]  
[checkbox/radio/grouppush]Pic_OFF[= picture]  
[checkbox/radio/grouppush]Pic_OFF_Pressed[= picture]  
[checkbox/radio/grouppush]Pic_ON[= picture]  
[checkbox/radio/grouppush]Pic_ON_Pressed[= picture]
```

Note that Picture properties with names ending "\_Pressed" are only used if the [DownPics](#) attribute is true.

### Arguments/Parameters

int iIndex	The index (starting from 0) of the picture
HBITMAP/HICON hNewPic	A handle to the new picture
WORD wType	Indicates the type of picture. Can be one of the three <a href="#">PictureType</a> values

### Return values

HBITMAP/HICON hPic	The handle of the requested picture
HBITMAP/HICON hOldPic	The handle of the previously set picture

### Remarks

C and C++ applications are responsible for destroying any pictures they add at run time.

At design time, the Picture attribute can be set from within the Dialog Editor, Resource Workshop, or Visual Basic, or picture resources can be specified in the [WindowText](#) directly.

### See Also

[AutoAdvance](#), [DownPics](#), [Palette](#)

## Alignment Codes

The Alignment codes are as follows:

<b><u>Code</u></b>	<b>Position in Client Area</b>
0	Top Left
1	Top Center
2	Top Right
3	Center Left
4	Center
5	Center Right
6	Bottom Left
7	Bottom Center
8	Bottom Right

## Relative Alignment Codes

The Relative Alignment codes are as follows:

<b><u>Code</u></b>	<b>Picture Relative to Text</b>
0	Above Left
1	Above Center
2	Above Right
3	Left
4	On Top
5	Right
6	Below Left
7	Below Center
8	Below Right

## Picture Types

<b><u>Code</u></b>	<b>Meaning</b>
HP_BITMAP	16 Color Bitmap
HP_DIB	256 Color Bitmap
HP_ICON	Icon

## Pressed Attribute

This attribute is set to TRUE while a button is depressed by the user pressing the mouse button or space bar. Setting this attribute displays the button as pressed or unpressed.

## Usage

## C

```
bIsPressed = SendMessage(hWnd, HBM_ISPRESSED, 0, 0L);  
SendMessage(hWnd, HBM_PRESS, (WPARAM)bPress, 0L);
```

## C++

```
bIsPressed = [CHButton.]IsPressed();  
[CHButton.]Press(bPress);
```

## VBX

Not Used

### Arguments/Parameters

BOOL bPressed

TRUE displays the button as pressed; FALSE, as unpressed

### Return values

BOOL bIsPressed

TRUE if button is being depressed

### Remarks

IsPressed returns TRUE only in the interval after the mouse has been clicked on a button but before it has been either released or moved from over the button. This attribute should not be used to test whether radiobuttons or checkboxes have been checked. To make these determinations, check the value of the State attribute.



## Quiet Attribute

When the control is in Quiet mode, it does not send notification messages to its parent. VBX controls will not fire events in Quiet mode.

### Usage

#### C

```
SendMessage(hWnd, HBM_BEQUIET, bValue, 0L);  
bQuiet = (BOOL)SendMessage(hWnd, HBM_ISQUIET, 0, 0L);
```

#### C++

```
[CHButton.]BeQuiet(bValue);  
bQuiet = [CHButton.]IsQuiet();
```

#### VBX

```
SendMessage(control(hWnd), HBM_BEQUIET, bValue, 0L)  
bQuiet = SendMessage(control(hWnd), HBM_ISQUIET, 0, 0L)  
See VBX Advanced Topics
```

### Arguments/Parameters

BOOL bValue	TRUE turns on Quiet mode, FALSE turns it off
-------------	--

### Return values

BOOL bIsQuiet	TRUE if the control is in Quiet mode
---------------	--------------------------------------

## RelAlign Attribute

A relative alignment code representing the alignment of the control's Picture with respect to its Text

### Usage

#### C

```
iRAlign = (int)SendMessage(hWnd, HBM_GETRALIGN, 0, 0L);  
SendMessage(hWnd, HBM_SETRALIGN, (WPARAM)iNewRAlign, 0L);
```

#### C++

```
iRAlign = [CHButtonObj.]GetRAlign();  
[CHButtonObj.]SetRAlign(iNewRAlign);
```

#### VBX

```
[form.][control.]AlignPicToText[= integer]
```

### Arguments/Parameters

int iNewRAlign	The new relative alignment code
----------------	---------------------------------

### Return values

int iRAlign	The current relative alignment code
-------------	-------------------------------------

### Remarks

The value of the RelAlign attribute is only used when PicAlign and TextAlign share the same value.

#### Relative Alignment Codes

At design time, the RelAlign attribute can be set from within the Dialog Editor, Resource Workshop, or Visual Basic, or can be specified in the WindowText directly.

### See Also

PicAlign, TextAlign

## Sound Attribute

The handle or filename of a sound resource that is played whenever the button is pressed. The SoundMode attribute determines whether the sound is played synchronously or asynchronously.

### Usage

#### C

```
hSound = (HANDLE)SendMessage(hWnd, HBM_GETSOUND, 0, 0L);  
SendMessage(hWnd, HBM_SETSOUND, (WPARAM)hNewSound, 0L);
```

#### C++

```
hSound = [CHButton.]GetSound();  
[CHButton.]SetSound(hNewSound);
```

#### VBX

```
[form.][control.]SoundFile[= filename]
```

### Arguments/Parameters

HANDLE hNewSound

The content of the handle depends on the setting of the sound mode. If the SoundMode is set to asynchronous, the handle must be a handle to a .WAV resource. Otherwise, it must be a resource information handle, as returned by FindResource.

### Return values

HANDLE hSound

The return value depends on the setting of the SoundMode. If the sound mode is set to asynchronous, the returned handle is the actual handle to a .WAV resource. Otherwise, it is a resource information handle, as returned by FindResource. To obtain the actual resource, call LoadResource.

### Remarks

The SoundFile property of the VBX button must always be set to a .WAV file name regardless of the SoundMode setting.

## SoundMode Attribute

Determines whether the sound is played synchronously -- regular execution is suspended until the sound terminates -- or asynchronously.

### Usage

#### C/C++

Window Style: **HBS\_ASYNC** *Sets the sound mode to asynchronous*

#### VBX

[*form.*][*control.*]**SoundMode**

### Remarks

The SoundMode property is read-only at run time.

## Squared Attribute

When set, the corners of a pushbutton are drawn square as opposed to rounded.

### Usage

#### C/C++

Window Style: HBS\_SQUARED

#### VBX

[*form.*][*control.*]**Squared**

### Remarks

This property is read-only at run time.

## State Attribute

The current state of the button in the sequence of conditions through which it cycles in its operation (e.g. "unchecked" and "checked" are States 0 and 1, respectively).

### Usage

#### C

```
iState = (int)SendMessage(hWnd, HBM_GETSTATE, 0, 0L);  
SendMessage(hWnd, HBM_SETSTATE, (WPARAM)iNewState, (LPARAM)bRedraw);  
iStateCount = (int)SendMessage(hWnd, HBM_GETSTATECOUNT, 0, 0L);
```

#### C++

```
iState = [CHButton.]GetState();  
[CHButton.]SetState(iState[, bRedraw = TRUE]);  
iStateCount = [CHButton.]GetStateCount();
```

#### VBX

```
[form.][control.]State[= iNewState]
```

### Arguments/Parameters

int iNewState	The new State value
BOOL bRedraw	TRUE forces immediate redraw

### Return values

int iState	The current State value
int iStateCount	The number of button states

### Remarks

For a button without the DownPics attribute set, the number of states is equal to the number of Pictures (Count attribute), or half this number if DownPics is set.

### See Also

AutoAdvance, Count, DownPics

## Text Attribute

A character string displayed on the button control. Line breaks can be inserted in the text using the ^ (carat) character. The button control will replace all carats with new line characters (\n).

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HBM_GETTEXT, (WPARAM) iMaxBytes,  
    (LPARAM) lpBuf);  
SendMessage(hWnd, HBM_SETTEXT, 0, (LPARAM) lpBuf);
```

#### C++

```
lBytesCopied = [CHButtonObj.]GetText(lpBuf [, iMaxBytes ==-1]);  
[CHButtonObj.]SetText(lpBuf);
```

#### VBX

```
[form.][control.]Text[= stringexpression]
```

### Arguments/Parameters

LPSTR lpBuf	A buffer for the control Text
int iMaxBytes	The maximum number of bytes to copy to lpBuf

### Return values

LONG lBytesCopied	The number of bytes actually copied to lpBuf
-------------------	--

### See Also

[TextAlign](#)

## TextAlign Attribute

An alignment code representing the position of the Text within the button's client area

### Usage

#### C

```
iTAlign = (int)SendMessage(hWnd, HBM_GETTALIGN, 0, 0L);  
SendMessage(hWnd, HBM_SETTALIGN, (WPARAM)iNewTAlign, 0L);
```

#### C++

```
iTAlign = [CHButton.]GetTAlign();  
[CHButton.]SetTAlign(iNewTAlign);
```

#### VBX

```
[form.][control.]AlignText[= integer]
```

### Arguments/Parameters

<code>int iNewPAlign</code>	The new alignment code
-----------------------------	------------------------

### Return values

<code>int iPAlign</code>	The current alignment code
--------------------------	----------------------------

### Remarks

[Alignment Codes](#)

At design time, the TextAlign attribute can be set from within the Dialog Editor, Resource Workshop, or Visual Basic, or can be specified in the [WindowText](#) directly.



## TextColor Attribute

The color of the control's Text.

### Usage

#### C

```
crTextColor = (COLORREF)SendMessage(hWnd, HBM_GETTEXTCOLOR, 0, 0L);  
SendMessage(hWnd, HBM_SETTEXTCOLOR, 0, crNewColor);
```

#### C++

```
crTextColor = [CHButton.]GetTextColor();  
[CHButton.]SetTextColor(crNewText);
```

#### VBX

```
[form.][control.]TextColor [= color]
```

### Arguments/Parameters

COLORREF crNewText	The new TextColor value
--------------------	-------------------------

### Return values

COLORREF crTextColor	The current TextColor value
----------------------	-----------------------------

### Remarks

At design time, the TextColor attribute can be set from within the Dialog Editor, Resource Workshop, or Visual Basic, or can be specified in the WindowText directly.

## TextIndent Attribute

When set, the button's Text is displayed with a white highlight to the lower left of each character, giving the text an indented appearance

### Usage

#### C/C++

Window Style: **HBS TEXTINDENT**

#### VBX

[*form.*][*control.*]**TextIndent**

### Remarks

This property is read-only at run time.

## TextJustify Attribute

Determines the justification of the button's Text if the text spans multiple lines. Text can be left, right or center (default) justified.

### Usage

#### C/C++

Window Styles:

HBS\_LJUST

HBS\_RJUST

#### VBX

[*form.*] [*control.*] **TextJustify**

### Remarks

This property is read-only at run time.

## Transparency Attribute

The button does not erase its background, allowing whatever is behind it to show through.

### Usage

#### C/C++

Window Style: **HBS\_TRANSPARENT**

#### VBX

[*form.*][*control.*]**Transparent**

### Remarks

When combined with the NoButton attribute, only the Picture and Text will appear, but the button will send Click and DbClick events for mouse clicks anywhere in its client area.

## HButt Events

Click

DbClick

### Click Event

Occurs whenever the user depresses a button using the mouse or space bar, or activates the button using a mnemonic, or activates a default pushbutton with the ENTER or ESCAPE keys.

#### Usage

**C/C++**

Notification code: **HBN\_CLICK**

**VBX**

**Sub** *ctlname\_Click* (Index **As Integer**)

#### Return Value

Not Used

## DbIcIck Event

Occurs whenever the user double-clicks on a button using the mouse.

### Usage

#### C/C++

Notification code: **HBN\_DBLCLICK**

#### VBX

**Sub** *ctlname\_DbIcIck* (Index **As Integer**)

### Return Value

Not Used

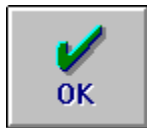
## HButton Window Text

This sample window text string is expanded below to show the meaning of each component:

```
441w; [DkBlue]OK; [0,255,0]OK1:OK2;Chimes
```

- 4**            The TextAlign alignment code.
- 4**            The PicAlign alignment code.
- 1**            The RelAlign relative alignment code is only used if the Text alignment and Picture alignment are the same.
- w**            The Data Type indicator is only used for CheckBox buttons.
- [DkBlue]**    The Text color in which the Text will be drawn.
- OK**            The Text that will appear on the button.
- [0,255,0]**    The Mask color can be used to make portions of a bitmap transparent.
- OK1:OK2**    Picture resource names must be separated by colons.
- Chimes**      Sound resource name or .WAV file name that will be played whenever the button is pressed.

This is the result:



## HComb, The WinWidgets ComboBox

- ▣ Attributes
- ▣ Methods
- ▣ Events

The HComb control is an aggregation of the HEdit and HList controls, providing a combination of their attributes, methods and events. Its purpose is to allow the user to select a choice from a list or type in a selection that is not listed. Common applications of the control include user extendible lists and filtered lists, in which the edit control's text is used to filter the list contents (e.g. the File Open dialog). The HComb Type, DropDownList, simply requires less screen space than a standard listbox.

A complete set of methods makes appending, inserting, deleting, selecting and retrieving items from the list as easy as possible.

HComb supports tab-expanded display, and sorting by Data or Codes. HComb has standard and 3D border styles and the ability to highlight itself upon gaining focus, helping users track their position on forms.

### Additional Topics

- ▣ Hot-Linking the ComboBox to your data
- ▣ Using HComb with the Visual Basic Data Control
- ▣ Using Codes in the ComboBox



## HComb Attributes

<u>Background</u>	<u>DataType</u>	<u>NonIntHeight</u>
<u>BorderStyle</u>	<u>DropHeight</u>	<u>Overwrite</u>
<u>Changed</u>	<u>EditData</u>	<u>Quiet</u>
<u>Code</u>	<u>EditMaxTextLen</u>	<u>Selection</u>
<u>CodeClass</u>	<u>EditScrollPos</u>	<u>SortMode</u>
<u>CodeLink</u>	<u>EditSelection</u>	<u>TabStops</u>
<u>CodeSize</u>	<u>EditText</u>	<u>Text</u>
<u>CodeType</u>	<u>EditTextLen</u>	<u>TextColor</u>
<u>Count</u>	<u>Font</u>	<u>TextLen</u>
<u>Data</u>	<u>Format</u>	<u>TopIndex</u>
<u>DataClass</u>	<u>HiliteBrush</u>	<u>Type</u>
<u>DataLink</u>	<u>HiliteOnFocus</u>	
<u>DataSize</u>	<u>Hunger</u>	

## Background Attribute

The color or pattern used to paint the background of the edit control

### Usage

#### C

```
hbrBkgnd = (HBRUSH) SendMessage(hWnd, HCM_GETBKGNDBRUSH, 0, 0L);  
hbrOldBkgnd = (HBRUSH) SendMessage(hWnd, HCM_SETBKGNDBRUSH,  
(WPARAM)hNewBrush, 0L);
```

#### C++

##### OWL

```
hbrBkgnd = [THCombObj.]GetBkgndBrush(void);  
hbrOldBkgnd = [THCombObj.]SetBkgndBrush(hbrNewBrush);
```

##### MFC

```
pBkgnd = [CHCombObj.]GetBkgndBrush(void);  
pOldBkgnd = [CHCombObj.]SetBkgndBrush(pNewBrush);
```

#### VBX

```
[form.][control.]BackColor[ = color ]
```

See *Visual Basic Language Reference*, "BackColor, ForeColor Properties"

### Arguments/Parameters

HBRUSH hbrNewBrush

Handle of a new brush to be set as the background brush

CBrush pNewBrush

Pointer to a CBrush object containing the new background brush handle

## Return values

HBRUSH hbrBkgnd	Handle to the current background brush
HBRUSH hbrOldBkgnd	Handle to the previous background brush
CBrush *pBkgnd	Pointer to a CBrush object containing the current background brush handle
CBrush *pOldBkgnd	Pointer to a CBrush object containing the previous background brush handle

## Remarks

C and C++ applications are responsible for destroying any brush they create.

# BorderStyle Attribute

## Usage

### C/C++

Window Styles:

HCS\_BORDER3D

HCS\_INDENT

HCS\_EXTRUDE

### VBX

[*form.*][*control.*]**BorderStyle**[= *None/Standard/Indented/Bump*]

## Remarks

BorderStyle can only be set at design time.

## Changed Attribute

A Boolean value indicating if the Data has been changed since it was last set

### Usage

#### C

```
bChanged = (BOOL)SendMessage(hWnd, HCM_HASCHANGED, 0, 0L);  
SendMessage(hWnd, HCM_SETCANGED, bVal, 0L);
```

#### C++

```
bChanged = [CHCombObj.]HasChanged();  
[CHCombObj.]SetChanged(bVal);
```

#### VBX

Not Used

### Arguments/Parameters

BOOL bVal

The new value for the Changed attribute

### Return Value

BOOL bChanged

TRUE if the Data has been changed since it was last set

### See Also

[Change event](#)

## Code Attribute

The native (binary) data maintained but not displayed by the control for each item in the List.

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HCM_GETCODE, (WPARAM) iIndex,  
    (LPARAM) lpCode);  
bSuccess = HCGetCode(hWnd, iIndex, lpCode);  
bSuccess = SendMessage(hWnd, HCM_SETCODE, (WPARAM) iIndex,  
    (LPARAM) lpCode);  
bSuccess = HCSetCode(hWnd, iIndex, lpCode);  
Get the code of item for the current Selection  
iResult = (int)SendMessage(hWnd, HCM_GETCURCODE, wSize, (LPARAM) lpBuf);
```

#### C++

```
bSuccess = [CHCombObj.]GetCode(iIndex, lpCode);  
bSuccess = [CHCombObj.]SetCode(iIndex, lpCode);  
Get the code of item for the current Selection  
iResult = [CHCombObj.]GetCurCode(lpBuf [, wSize = -1]);
```

#### VBX

```
[form.][control.]Code(iIndex) [= stringexpression]
```

The **Code** attribute is a string array; *iIndex* is a required parameter

### Arguments/Parameters

int iIndex	The index of the item
void FAR *lpCode	Pointer to a buffer for the code
WORD wSize	Maximum number of bytes to copy (used only for strings).

### Return values

BOOL bSuccess	TRUE if the operation is a success
int iBytesCopied	Number of bytes copied.

### Remarks

The code for an item cannot be set for a list that is sorted by codes.  
Combo boxes with associated edit controls cannot have codes.

### See Also

[CodeClass](#), [CodeLink](#), [CodeSize](#), [CodeType](#), [Data](#)

## CodeClass Attribute

One of the data classes defined in the [Data Engine](#) chapter.

### Usage

#### C

```
cCodeClass = (char)SendMessage(hWnd, HCM_GETCODECLASS, 0, 0L);
```

#### C++

```
cCodeClass = [CHCombObj.]GetCodeClass();
```

#### VBX

```
[form.][control.]CodeClass
```

### Return values

char cCodeClass                      One of the [data class character codes](#)

### Remarks

The CodeClass attribute can only be set at design time. Control over this attribute is provided by the control design dialogs in Dialog Editor and Resource Workshop, and by the Properties dialog in Visual Basic. When creating windows dynamically in C and C++, the DataClass is included in the [WindowText](#).

### See Also

[Code](#), [CodeType](#)

## CodeLink Attribute

A pointer to the variable or buffer that will be updated with the new item's code when the Selection changed

### Usage

#### C

```
lpCodeLink = (void FAR *)SendMessage(hWnd, HCM_GETCODELINK, 0, 0L);  
lBytesCopied = (LONG)SendMessage(hWnd, HCM_SETCODELINK, (WPARAM)bSelect,  
(LPARAM)(LPVOID)lpNewLink);  
HCSetCodeLink(hWnd, lpCode, bSelect);
```

#### C++

```
lpCodeLink = [CHCombObj.]GetCodeLink();  
bSuccess = [CHCombObj.]SetCodeLink(lpNewLink [, bSelect = TRUE]);
```

#### VBX

Not Used

### Arguments/Parameters

void FAR \*lpNewLink  
BOOL bSelect

Pointer to program data  
If TRUE, selection will be set to match the contents of lpNewLink

### Return values

void FAR \*lpCodeLink  
  
BOOL bSuccess  
LONG lBytesCopied

Pointer to the current CodeLink. NULL if no CodeLink has been set.  
TRUE if CodeLink was set successfully  
Number of bytes copied.

### See Also

[Code](#)

## CodeSize Attribute

The size of each item's Code in bytes.

### Usage

#### C

```
iCodeSize = SendMessage(hWnd, HCM_GETCODESIZE, (WPARAM)iIndex, 0L);
```

#### C++

```
iCodeSize = [CHCombObj.]GetCodeSize([iIndex = -1]);
```

#### VBX

Not Used

### Arguments/Parameters

`int iIndex`

The index of an item--only necessary with character string CodeTypes.

### Return values

`int iCodeSize`

The size of the Code

### Remarks

CodeSize may be variable only for NULL-terminated strings.

### See Also

[CodeType](#)



## CodeType Attribute

One of the data types defined in the [DataEngine](#) chapter.

### Usage

#### C

```
cCodeType = (char)SendMessage(hWnd, HCM_GETCODETYPE, 0, 0L);
```

#### C++

```
cCodeType = [CHCombObj.]GetCodeType();
```

#### VBX

```
[form.][control.]CodeType
```

### Return values

char cCodeType

One of the [data type character codes](#)

### See Also

[Code](#), [CodeClass](#), [CodeSize](#)

## Count Attribute

The number of items in the List

### Usage

#### C

```
iCount = SendMessage(hWnd, HCM_GETCOUNT, 0, 0L);
```

#### C++

```
iCount = [CHCombObj.]GetCount();
```

#### VBX

```
[form.][control.]Count
```

### Return values

int iCount

The current number of items in the list

## Data Attribute

The native (binary) data maintained and displayed by the control for each item in the List.

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HCM_GETDATA, (WPARAM) iIndex,  
    (LPARAM) lpData);
```

```
bSuccess = HCGetData(hWnd, iIndex, lpData);
```

To get the data of the currently selected item:

```
iBytesCopied = (int)SendMessage(hWnd, HCM_GETCURDATA, wSize,  
    (LPARAM) lpBuf);
```

#### C++

```
bSuccess = [CHCombObj.]GetData(iIndex, lpData);
```

To get the data of the currently selected item:

```
iBytesCopied = [CHCombObj.]GetCurData(lpBuf [, wSize = -1]);
```

#### VBX

```
[form.][control.]Data(iIndex)
```

The **Data** attribute is a string array; *iIndex* is a required parameter

### Arguments/Parameters

int iIndex	The index of the item
void FAR *lpData	Pointer to a buffer for the Data
WORD wSize	Maximum number of bytes to copy (used only for strings).

### Return values

BOOL bSuccess	TRUE if the operation is a success
int iBytesCopied	Number of bytes copied.

### See Also

[Code](#), [DataClass](#), [DataLink](#), [DataSize](#), [DataType](#)

## DataClass Attribute

One of the data classes defined in the [Data Engine](#) chapter.

### Usage

#### C

```
cDataClass = (char)SendMessage(hWnd, HCM_GETDATACLASS, 0, 0L);
```

#### C++

```
cDataClass = [CHCombObj.]GetDataClass();
```

#### VBX

```
[form.][control.]DataClass
```

### Return values

char cDataClass                      One of the [data class character codes](#)

### Remarks

The DataClass attribute can only be set at design time. Control over this attribute is provided by the control design dialogs in Dialog Editor and Resource Workshop, and by the Properties dialog in Visual Basic. When creating windows dynamically in C and C++, the DataClass is included in the [WindowText](#).

### See Also

[Data](#), [DataType](#)

## DataLink Attribute

A pointer to the variable or buffer that will be updated with the new item's Data when the Selection changed

### Usage

#### C

```
lpDataLink = (void FAR *)SendMessage(hWnd, HCM_GETDATALINK, 0, 0L);  
lBytesCopied = (LONG)SendMessage(hWnd, HCM_SETDATALINK, (WPARAM)bSelect,  
(LPARAM)(LPVOID)lpNewLink);  
HCSetDataLink(hWnd, lpData, bSelect);
```

#### C++

```
lpDataLink = [CHCombObj.]GetDataLink();  
bSuccess = [CHCombObj.]SetDataLink(lpNewLink [, bSelect = TRUE]);
```

#### VBX

Not Used

### Arguments/Parameters

void FAR *lpNewLink	Pointer to program data
BOOL bSelect	If TRUE, selection will be set to match the contents of lpNewLink

### Return values

void FAR *lpDataLink	Pointer to the current DataLink. NULL if no DataLink has been set.
BOOL bSuccess	TRUE if DataLink was set successfully

### Remarks

### See Also

[Data](#)

## DataSize Attribute

The size of each item's Data in bytes.

### Usage

#### C

```
iDataSize = SendMessage(hWnd, HCM_GETDATASIZE, (WPARAM)iIndex, 0L);
```

#### C++

```
iDataSize = [CHCombObj.]GetDataSize([iIndex = -1]);
```

#### VBX

Not Used

### Arguments/Parameters

<code>int iIndex</code>	The index of an item--only necessary with character string DataTypes.
-------------------------	---

### Return values

<code>int iDataSize</code>	The size of the Data
----------------------------	----------------------

### Remarks

DataSize may be variable only for NULL-terminated strings.

### See Also

[DataType](#)

## DataType Attribute

One of the data types defined in the [DataEngine](#) chapter.

### Usage

#### C

```
cDataType = (char)SendMessage(hWnd, HCM_GETDATATYPE, 0, 0L);
```

#### C++

```
cDataType = [CHCombObj.]GetDataType();
```

#### VBX

```
[form.][control.]DataType
```

### Return values

char cDataType

One of the [data type character codes](#)

### See Also

[Data](#), [DataClass](#), [DataSize](#)

## DropHeight Attribute

The height of the drop-down portion of a combobox.

### Usage

#### C

```
iDropHeight = (int)SendMessage(hWnd, HCM_GETDROPHEIGHT, 0, 0L);  
bSuccess = (BOOL)SendMessage(hWnd, HCM_SETDROPHEIGHT,  
(WPARAM)iNewHeight, 0L);
```

#### C++

```
iDropHeight = [CHCombObj.]GetDropHeight();  
bSuccess = [CHCombObj.]SetDropHeight(iNewHeight);
```

#### VBX

```
[form.][control.]DropHeight[= integer]
```

### Arguments/Parameters

<code>int iNewHeight</code>	The new DropHeight
-----------------------------	--------------------

### Return values

<code>int iDropHeight</code>	The current DropHeight value
------------------------------	------------------------------



## EditData Attribute

The native (binary) data displayed by the control's edit box, of a type defined in the [Data Engine](#) chapter of this manual.

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HCM_GETEDITDATA, (WPARAM) iMaxBytes,  
(LPARAM) lpData);  
lBytesCopied = SendMessage(hWnd, HCM_SETEDITDATA, 0, (LPARAM) lpData);
```

#### C++

```
int iBytesCopied = [CHCombObj.]GetEditData(lpData [, iMaxBytes = 0] );  
int iBytesCopied = [CHCombObj.]SetEditData(lpData);
```

#### VBX

Not Used

### Arguments/Parameters

void FAR *lpData	Pointer to data
int iMaxBytes	Maximum number of bytes to copy (used only for strings)

### Return values

LONG lBytesCopied	Number of bytes actually copied to or from the control
-------------------	--

### Remarks

Due to Visual Basic's flexible type handling, control data for VBX controls can be set and retrieved via the [EditText](#) property.

[Drop-down list](#) boxes have no edit control associated with them and thus do not have an EditData attribute

### See Also

[DataClassHComb\\_Attr\\_DataClass](#), [DataLinkHComb\\_Attr\\_DataLink](#),  
[DataTypeHComb\\_Attr\\_DataType](#)

## EditMaxTextLen Attribute

The maximum number of characters that can be entered into the edit box of a control with the HC\_STRING DataClass

### Usage

#### C

```
iMaxLen = (int)SendMessage(hWnd, HCM_GETEDITMAXTEXTLEN, 0, 0L);  
SendMessage(hWnd, HCM_SETEDITMAXTEXTLEN, (WPARAM)iLen, 0L);
```

#### C++

```
iMaxLen = [CHCombObj.]GetMaxTextLen();  
[CHCombObj.]SetMaxTextLen(iLen);
```

#### VBX

```
[form.][control.]EditMaxText[= iLen]
```

### Arguments/Parameters

<code>int iLen</code>	New maximum length. A value of -1 removes the maximum text length
-----------------------	---

### Return values

<code>int iMaxLen</code>	The currently set maximum text length.
--------------------------	--

### Remarks

The MaximumTextLength can only be set for the HC\_STRING DataClass. Maximum lengths for other classes are determined by their Format strings.

## EditScrollPos Attribute

The number of characters that have been scrolled out of the control's edit boxes client area

### Usage

#### C

```
iScrollPos = (int)SendMessage(hWnd, HCM_GETEDITSCROLLPOS, 0, 0L);  
iScrollPos = SendMessage(hWnd, HCM_SETEDITSCROLLPOS, (WPARAM)iScroll,  
(LPARAM)bRedraw);
```

#### C++

```
iScrollPos = [CHCombObj.]GetEditScrollPos();  
iScrollPos = [CHCombObj.]SetEditScrollPos(iScroll [, bRedraw = TRUE]);
```

#### VBX

Not Used

### Arguments/Parameters

int iScroll

The number of characters to scroll off the left side for left justified text and off the right side for right justified text

BOOL bRedraw

A value of TRUE causes the control to repaint itself immediately

### Return values

int iScrollPos

The number of characters scrolled off the left side for left justified text and off the right side for right justified text

## EditSelection Attribute

The text within the control's edit box that is currently selected

### Usage

#### C

```
lSel = SendMessage(hWnd, HCM_GETEDITSEL, 0, 0L);  
SendMessage(hWnd, HCM_SETEDITSEL, 0, (LPARAM)lNewSel);
```

#### C++

```
lSel = [CHCombObj.]GetEditSel();  
[CHCombObj.]SetEditSel(lNewSel);
```

#### VBX

```
[form.][control.]EditSelLen [= length]  
[form.][control.]EditSelStart [= index]  
[form.][control.]EditSelText [= stringexpression]  
See Visual Basic Language Reference, "SelLength, SelStart, SelText Properties"
```

### Arguments/Parameters

LONG lNewSel

Contains the starting position in the low-order word and the character position of the first non-selected character after the selection in the high-order word

### Return values

LONG lSel

Contains the starting position in the low-order word and the character position of the first non-selected character after the selection in the high-order word

### Remarks

Visual Basic programmers can replace the EditSelection by assigning a new value to the EditSelText property

## EditText Attribute

A character string representing formatted [EditData](#)

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HCM_GETEDITTEXT, (WPARAM) iMaxBytes,  
(LPARAM) lpBuf);
```

#### C++

```
lBytesCopied = [CHCombObj].GetEditText(lpBuf [, iMaxBytes = -1]);
```

#### VBX

```
[form]. [control].Text
```

### Arguments/Parameters

LPSTR lpBuf

A buffer for the control Text

int iMaxBytes

The maximum number of bytes to copy to lpBuf

### Return values

LONG lBytesCopied

The number of bytes actually copied to lpBuf

### Remarks

[Drop-down list](#) comboboxes have no edit control associated with them and thus do not have an EditText attribute

### See Also

[EditTextLength](#)

## EditTextLength Attribute

The length in characters of the editable Text in the combobox

### Usage

#### C

```
iEditTextLen = SendMessage(hWnd, HCM_GETEDITTEXTLEN, 0, 0L);
```

#### C++

```
iEditTextLen = GetEditTextLen();
```

#### VBX

```
Len([control.]Text)
```

### Return values

<code>int iEditTextLen</code>	The current EditTextLength
-------------------------------	----------------------------

### See Also

[EditMaxTextLen](#), [EditText](#)

## Font Attribute

The font used by the control

### Usage

#### C

```
hfFont = (HFONT)SendMessage(hWnd, HCM_GETFONT, 0, 0L);  
hfOldFont = SendMessage(hWnd, HCM_SETFONT, (WPARAM)hfNewFont,  
(LPARAM)bRedraw);
```

#### C++

##### OWL

```
hfFont = [THCombObj.]GetFont();  
hfOldFont[THCombObj.]SetFont(hfNewFont [, bRedraw = TRUE] );
```

##### MFC

```
pFont = [CHCombObj.]GetFont();  
pOldFont[CHCombObj.]SetFont(pNewFont [, bRedraw = TRUE] );
```

#### VBX

```
[form.][control.]FontBold[= boolean]  
[form.][control.]FontItalic[= boolean]  
[form.][control.]FontName[= font]  
[form.][control.]FontSize[= points]  
[form.][control.]FontStrikethru[= boolean]  
[form.][control.]FontUnderline[= boolean]
```

See Visual Basic Language Reference, "FontName Property"

### Arguments/Parameters

HFONT hfNewFont	Handle of the font to be set
CFont *pFont	Pointer to a CFont object containing the handle to the font to be set
BOOL bRedraw	A value of TRUE causes the control to repaint immediately

### Return values

HFONT hfFont	Handle to the control's current font
HFONT hfOldFont	Handle to the control's previous font
CFont *hfFont	Pointer to a CFont object containing the handle to the control's current font
CFont *fOldFont	Pointer to a CFont object containing the handle to the control's previous font

### Remarks

C and C++ applications are responsible for destroying any fonts they create.

## Format Attribute

A NULL-terminated character string that describes the way the Data is to be displayed

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HCM_GETFORMAT, (WPARAM)iMaxBytes,  
(LPARAM)lpstrBuf);  
lBytesCopied = SendMessage(hWnd, HCM_SETFORMAT, (WPARAM)bRedraw,  
(LPARAM)lpstrBuf);
```

#### C++

```
lBytesCopied = [CHCombObj.]GetFormat(lpstrBuf [, iMaxBytes = -1] );  
lBytesCopied = [CHCombObj.]SetFormat(lpstrBuf [, bRedraw = TRUE] );
```

#### VBX

```
[form.][control.]FormatString[= string]
```

### Arguments/Parameters

LPSTR lpstrBuf	Buffer that contains a new format string or will receive the existing one
int iMaxBytes	The maximum bytes to copy to the buffer. A value of -1 copies the entire format string.
BOOL bRedraw	A value of TRUE causes the control to redraw immediately.

### Return values

LONG lBytesCopied	The number of bytes actually copied to or from the buffer
-------------------	---

### Remarks

An initial format string is contained in the [WindowText](#) when a control is created.



## HiliteBrush Attribute

The color or pattern used to paint the background when the control receives focus

### Usage

#### C

```
hbrHilite = SendMessage(hWnd, HCM_GETHILITEBRUSH, 0, 0L);  
hbrOldHilite = SendMessage(hWnd, HCM_SETHILITEBRUSH,  
(WPARAM)hbrNewHilite, 0L);
```

#### C++

##### OWL

```
hbrHilite = [THCombObj.]GetHiliteBrush();  
hbrOldHilite = [THCombObj.]SetHiliteBrush(hbrNewHilite);
```

##### MFC

```
pHilite = [CHCombObj.]GetHiliteBrush();  
pOldHilite = [CHCombObj.]SetHiliteBrush(pNewHilite);
```

#### VBX

```
[form.][control.]HiliteColor[= color]
```

### Arguments/Parameters

HBRUSH hbrNewHilite	Handle of the new brush
CBrush *pNewHilite	Pointer to a CBrush object containing the handle of the new brush

### Return values

HBRUSH hbrHilite	Handle of the control's current brush
HBRUSH hbrOldHilite	Handle of the control's previous brush
CBrush *pHilite	Pointer to a CBrush object containing the handle of the current brush
CBrush *pOldHilite	Pointer to a CBrush object containing the handle of the previous brush

### Remarks

## HiliteOnFocus Attribute

When set, the HiliteBrush is used to paint the background when the control receives input focus. If no HiliteBrush is selected, the control uses a white brush.

### Usage

#### C/C++

Window Style: HCS\_HILITE

#### VBX

[*form.*] [*control.*] **HiliteOnFocus**

### Remarks

HiliteOnFocus is read-only at run time

## Hunger Attribute

When set, the control swallows *Enter* and *Esc* keyboard messages and notifies its parent.

### Usage

#### C/C++

Window Style: HCS\_HUNGER

#### VBX

[*form.*] [*control.*] **Hunger**

### Remarks

The Hunger attribute is obsolete and is included here for backward compatibility. We recommend that C and C++ programmers use a Filter Procedure or Dynamic Subclassing, respectively, to implement this functionality. Hunger is read-only at run time.

## NonIntHeight Attribute

When set, the control can display a partial item at the bottom of the list.

### Usage

#### C/C++

Window Style: HCS\_NONINHEIGHT

#### VBX

[*form.*] [*control.*]NonIntHeight

### Remarks

NonIntHeight can only be set at design time

## Overwrite Attribute

Determines whether text overwrites existing text or is inserted as it is input into a control of the HC\_STRING DataClass.

### Usage

#### C

```
bOverwrite = (BOOL)SendMessage(hWnd, HCM_GETOVERWRITEMODE, 0, 0L);  
SendMessage(hWnd, HCM_SETOVERWRITEMODE, (WPARAM)bMode, 0L);
```

#### C++

```
bOverwrite = [CHCombObj.]GetOverwriteMode();  
[CHCombObj.]SetOverwriteMode( [bMode = TRUE] );
```

#### VBX

```
[form.][control.]OverwriteMode[= bMode]
```

### Arguments/Parameters

BOOL bMode

TRUE for overwrite, FALSE for insert

### Return values

BOOL bOverwrite

Current mode of the control, TRUE for overwrite, FALSE for insert

### Remarks

Overwrite mode is only used with the HC\_STRING DataClass. Other DataClasses insert or overwrite characters based on the position of the caret in the editing template.

## Quiet Attribute

When the control is in Quiet mode, it does not send notification messages to its parent. VBX controls will not fire events in Quiet mode.

### Usage

#### C

```
SendMessage(hWnd, HCM_BEQUIET, bValue, 0L);  
bQuiet = (BOOL)SendMessage(hWnd, HCM_ISQUIET, 0, 0L);
```

#### C++

```
[CHCombObj.]BeQuiet(bValue);  
bQuiet = [CHCombObj.]IsQuiet();
```

#### VBX

```
SendMessage(control.hWnd, HCM_BEQUIET, bValue, 0L)  
bQuiet = SendMessage(control.hWnd, HCM_ISQUIET, 0, 0L)  
See VBX Advanced Topics
```

### Arguments/Parameters

BOOL bValue	TRUE turns on Quiet mode, FALSE turns it off
-------------	--

### Return Value

BOOL bQuiet	TRUE if control is in Quiet Mode
-------------	----------------------------------

## Selection Attribute

The index of the currently selected item

### Usage

#### C

```
iCurSel = (int)SendMessage(hWnd, HCM_GETCURSEL, 0, 0L);  
iResult = (int)SendMessage(hWnd, HCM_SETCURSEL, (WPARAM)iIndex, 0L);
```

#### C++

```
iCurSel = [CHCombObj.]GetCurSel();  
iResult = [CHCombObj.]SetCurSel(iIndex);  
bSelected = [CHCombObj.]IsSelected(iIndex);
```

#### VBX

```
[form.][control.]Selection[= index]
```

### Arguments/Parameters

`int iIndex`

The index of the item to Select

### Return values

`int iCurSel`

The index of the current Selection

`int iResult`

HLERR\_NOTFOUND if Selection cannot be set.

`BOOL bSelected`

TRUE if the item at `iIndex` is selected

### See Also

[Code](#), [Data](#)

## SortMode Attribute

Determines how items in the control's list are to be sorted

### Usage

#### C/C++

Window Styles:

HCS\_SORTBYDATA

HCS\_SORTBYCODE

#### VBX

`[form.][control.]SortMode [= None/ByData/ByCodes]`

### Remarks

Sort can only be set at design time.



## TabStops Attribute

An array of integer tabstops representing spacing in characters

### Usage

#### C

Window Style: **HCS\_USETABS**

```
bSuccess = (BOOL)SendMessage(hWnd, HCM_SETTABSTOPS, (WPARAM) iNumber,  
(LPARAM) lpTabs);
```

#### C++

Window Style: **HCS\_USETABS**

```
bSuccess = [CHCombObj.]SetTabStops(iNumber, lpTabs);
```

#### VBX

Not Used

### Arguments/Parameters

int iNumber

The number of tab stops to set

int far \*lpTabs

An array of integer TabStops in dialog units

### Return values

BOOL bSuccess

TRUE if TabStops were set correctly

### Remarks

If iNumber is zero and lParam is NULL, the default tab stop is eight dialog units.

If iNumber is 1, the TabStops are spaced evenly based on the first value pointed to by the lParam.

To set and display tabs, the HCS\_USETABS style or UseTabs property must be set at design time.

## Text Attribute

A character string representing the formatted data for each item

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HCM_GETTEXT, iIndex, lpBuf);
```

#### C++

```
lBytesCopied = [CHCombObj.]GetText(lpBuf, iIndex);
```

#### VBX

```
[form.][control.]Text
```

### Arguments/Parameters

int iIndex

The index of an item in the List

LPSTR lpBuf

A buffer for the Text

### Return values

lBytesCopied

The actual number of bytes copied to lpBuf

### See Also

[TextColor](#), [TextLen](#)

## TextColor Attribute

The color used when painting the Text

### Usage

#### C

```
crTextColor = (COLORREF)SendMessage(hWnd, HCM_GETTEXTCOLOR, bNegative, 0L);  
SendMessage(hWnd, HCM_SETTEXTCOLOR, bNegative, crNewColor);
```

#### C++

```
crTextColor = [CHCombObj.]GetTextColor([bNegative = FALSE]);  
[CHCombObj.]SetTextColor(crNewColor [, bNegative = FALSE]);
```

#### VBX

```
[form.][control.]TextColor[= color]  
[form.][control.]TextColor_Neg[= color]
```

### Arguments/Parameters

BOOL bNegative	If TRUE, the TextColor for negative numbers is gotten or set.
COLORREF crNewColor	The new TextColor

### Return values

COLORREF crTextColor	The current TextColor
----------------------	-----------------------

### Remarks

The TextColor for can be set at design time or as part of the [WindowText](#)

### See Also

[Text](#), [Format](#)

## TextLen Attribute

The length of the Text in characters

### Usage

#### C

```
iTextLen = SendMessage(hWnd, HCM_GETTEXTLEN, (WPARAM)iIndex, 0L);
```

#### C++

```
iTextLen = [CHCombObj.]GetTextLen(iIndex);
```

#### VBX

```
Len([control.]Text)
```

### Arguments/Parameters

int iIndex

The index of an item in the List

### Return values

int iTextLen

The TextLength for item iIndex

### See Also

[Text](#)

## TopIndex Attribute

The index of the item displayed at the top of the List

### Usage

#### C

```
iTop = (int)SendMessage(hWnd, HCM_GETTOPINDEX, 0, 0L);  
iTop = (int)SendMessage(hWnd, HCM_SETTOPINDEX, (WPARAM)iIndex, 0L);
```

#### C++

```
iTop = [CHCombObj.]GetTopIndex();  
iTop = [CHCombObj.]SetTopIndex(iIndex);
```

#### VBX

```
[form.][control.]TopIndex[= index]
```

### Arguments/Parameters

<code>int iIndex</code>	The index of the new top item
-------------------------	-------------------------------

### Return values

<code>int iTop</code>	The index of the top item
-----------------------	---------------------------

## Type Attribute

The type of combobox. The *Simple* ComboBox is an edit control above a listbox. A *DropDown* ComboBox is an edit control with an arrow button that pops up a listbox when pressed. A *DropList* ComboBox is a single-line listbox with an arrow button that pops up a multi-line listbox when pressed.

## Usage

### C/C++

Window Styles:

HCS\_HASEDIT

HCS\_DROPDOWN

### VBX

[*form.*][*control.*]**Type**[= *Simple/DropList/DropDown*]

## Remarks

Type can only be set at design time

## HComb Methods

<u>Add</u>	<u>Insert</u>	<u>SelectString</u>
<u>Delete</u>	<u>Reset</u>	
<u>FindCode</u>	<u>Retrieve</u>	
<u>FindData</u>	<u>SelectCode</u>	
<u>FindString</u>	<u>SelectData</u>	

## Add Method

Adds an item or items to a List. If the list is unsorted, addition occurs at the end of the list.

### Usage

#### C

Add a single item

```
iNewIndex = HCAddItem(hWnd, lpData);
```

Add a single item with code

```
iNewIndex = HCAddItemEx(hWnd, lpData, lpCode);
```

Add multiple items

```
iNumber = HCAddItems(hWnd, iCount, lpData);
```

Add multiple items with codes

```
iNumber = HCAddItemsEx(hWnd, iCount, lpData, lpCode);
```

#### C++

Add a single item

```
iNewIndex = [CHCombObj].AddItem(lpData);
```

Add a single item with code

```
iNewIndex = [CHCombObj].AddItemEx(lpData, lpCode);
```

Add multiple items

```
iNumber = [CHCombObj].AddItems(iCount, lpData);
```

Add multiple items with codes

```
iNumber = [CHCombObj].AddItemsEx(iCount, lpData, lpCode);
```

#### VBX

Add a single item

```
[form].[control].AddItem strData
```

Add a single item with code

```
iNewIndex = VCAddItemEx(control.hWnd, strData, strCode)
```

### Arguments/Parameters

int iCount

The number of items to add

void FAR \*lpData

A pointer to the Data item (single) or array (multiple)

strData  
void FAR \*lpCode  
  
strCode

A string representing the Data to add  
A pointer to the Code item (single) or array  
(multiple)  
A string representing the Code to add

### **Return values**

int iNewIndex  
  
int iNumber

The index at which an item was added or an error code  
The number of items successfully added or an error code

### **See Also**

[Insert](#)



## Delete Method

Deletes an item or items from the List

### Usage

#### C

Delete a single item

```
bSuccess = (BOOL)SendMessage(hWnd, HCM_DELETEITEM, (WPARAM)iIndex, 0L);
```

Delete multiple items

```
iNumber = HCDeleteItems(hWnd, wSearchCat, iCount, lpSearchInfo);
```

#### C++

Delete a single item

```
bSuccess = [CHCombObj.]DeleteItem(iIndex);
```

Delete multiple items

```
iNumber = [CHCombObj.]DeleteItems(wSearchCat, iCount, lpSearchInfo);
```

#### VBX

```
[form.][control.]RemoveItem iIndex
```

### Arguments/Parameters

<code>int iIndex</code>	The index of an item in the List
<code>WORD wSearchCat</code>	A <u>search category</u>
<code>int iCount</code>	The number of items to find and delete
<code>void far *lpSearchInfo</code>	A pointer to an array of Data items, Codes, or Indices depending on the value of wSearchCat

### Return values

<code>BOOL bSuccess</code>	TRUE if the item was deleted successfully
<code>int iNumber</code>	The number of items successfully deleted

### See Also

Add, Insert

## FindCode Method

Gets the index of an item given its code

### Usage

#### C

```
iIndex = (int)SendMessage(hWnd, HCM_FINDCODE, (WPARAM)iStart,  
(LPARAM)lpCode);
```

#### C++

```
iIndex = [CHCombObj.]FindCode(iStart, lpCode);
```

#### VBX

```
iIndex = VCFindCode(control.hWnd, iStart, strCode)
```

### Arguments/Parameters

`int iStart`

The index at which to begin searching

`void FAR *lpCode`

A pointer to the Code to search for

`strCode`

A string representing the Code to search for

### Return values

`int iIndex`

The index of the item or HCERR\_NOTFOUND if no match was found

### See Also

[FindData](#), [FindString](#)

## FindData Method

Gets the index of an item given its Data

### Usage

#### C

```
iIndex = (int)SendMessage(hWnd, HCM_FINDDATA, (WPARAM)iStart,  
(LPARAM)lpData);
```

#### C++

```
iIndex = [CHCombObj.]FindData(iStart, lpData);
```

#### VBX

```
iIndex = VCFindData (control.hWnd, iStart, strData)
```

### Arguments/Parameters

`int iStart`

The index at which to begin searching

`void FAR *lpData`

A pointer to the Data to search for

`strData`

A string representing the Data to search for

### Return values

`int iIndex`

The index of the item or HCERR\_NOTFOUND if no match was found

### See Also

[FindCode](#), [FindString](#)

## FindString Method

Returns the index of an item given some or all of its Text. FindString is not case-sensitive.

### Usage

#### C

```
iIndex = (int)SendMessage(hWnd, HCM_FINDSTRING, (WPARAM)iStart,  
(LPARAM)lpText);
```

#### C++

```
iIndex = [CHCCombObj.]FindString(iStart, lpText);
```

#### VBX

```
iIndex = VCFindString (control.hWnd, iStart, strText)
```

### Arguments/Parameters

<code>int iStart</code>	The index at which to begin searching
<code>void FAR *lpText</code>	A pointer to the <u>Text</u> to search for
<code>strText</code>	A string representing the <u>Text</u> to search for

### Return values

<code>int iIndex</code>	The index of the item or HCERR_NOTFOUND if no match was found
-------------------------	---

### Remarks

FindString will attempt to find the closest match when passed a partial string.

### See Also

[FindCode](#), [FindData](#)

## Insert Method

Inserts an item or items into an unsorted List

### Usage

#### C

Insert a single item

```
iNewIndex = HCInsertItem(hWnd, iPos, lpData);
```

Insert a single item with code

```
iNewIndex = HCInsertItemEx(hWnd, iPos, lpData, lpCode);
```

Insert multiple items

```
iNumber = HCInsertItems(hWnd, iPos, iCount, lpData);
```

Insert multiple items with codes

```
iNumber = HCInsertItemsEx(hWnd, iPos, iCount, lpData, lpCode);
```

#### C++

Insert a single item

```
iNewIndex = [CHCombObj.]InsertItem(iPos, lpData);
```

Insert a single item with code

```
iNewIndex = [CHCombObj.]InsertItemEx(iPos, lpData, lpCode);
```

Insert multiple items

```
iNumber = [CHCombObj.]InsertItems(iPos, iCount, lpData);
```

Insert multiple items with codes

```
iNumber = [CHCombObj.]InsertItemsEx(iPos, iCount, lpData, lpCode);
```

#### VBX

Insert a single item

```
[form.][control.]AddItem strData, iPos
```

Insert a single item with code

```
iNewIndex = VCInsertItemEx(control.hWnd, iPos, strData, strCode)
```

### Arguments/Parameters

<code>int iPos</code>	The index at which to insert
<code>int iCount</code>	The number of items to insert
<code>void FAR *lpData</code>	A pointer to the Data item (single) or array (multiple)
<code>strData</code>	A string representing the Data to insert
<code>void FAR *lpCode</code>	A pointer to the Code item (single) or array (multiple)
<code>strCode</code>	A string representing the Code to insert

### Return values

<code>int iNewIndex</code>	The index at which an item was inserted or an <u>error code</u>
<code>int iNumber</code>	The number of items successfully inserted or an

error code

**See Also**  
Add

## Reset Method

Removes all items from the List

### Usage

#### C

```
SendMessage(hWnd, HCM_RESETCONTENT, 0, 0L);  
bSuccess = HCEmptyList(hWnd);
```

#### C++

```
bSuccess = [CHCombObj.]EmptyList();
```

#### VBX

```
[form.][control.]Clear
```

### Return values

BOOL bSuccess

TRUE if the List was emptied successfully

## Retrieve Method

Gets the Data items, Codes, or indices based on a search criterion

### Usage

#### C

```
iNumber = HCGetItems(hWnd, iCount, wReturnCat, lpReturnInfo, wSearchCat,  
lpSearchInfo)
```

#### C++

```
iNumber = [CHCombObj.]GetItems(iCount, wReturnCat, lpReturnInfo,  
wSearchCat, lpSearchInfo)
```

#### VBX

Not Used

### Arguments/Parameters

```
int iCount  
WORD wReturnCat  
void FAR *lpReturnInfo
```

```
WORD wSearchCat  
void FAR *lpSearchInfo
```

The maximum number of items to find and return

A return category

A pointer to an array of Data items, Codes or Indices, depending on the wReturnCategory to receive the returned data.

A search category

A pointer to an array of Data items, Codes or Indices, depending on the wSearchCategory.

### Return values

```
int iNumber
```

The actual number of items returned

### See Also

[Selection](#)



## SelectCode Method

Selects an item based on its Code and performs an action

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HCM_SELECTCODE, wAction, (LPARAM)lpCode);  
HCSelctCode(wAction, lpCode);
```

#### C++

```
bSuccess = [CHCombObj.]SelectCode(lpCode, wAction);
```

#### VBX

```
[form.][control.]SelectedCode[ = strCode]
```

### Arguments/Parameters

WORD wAction	An <u>action code</u>
void FAR *lpCode	A pointer to the Code to match
strCode	A string representing the Code to match

### Return Value

BOOL bSuccess	TRUE if selection is successful
---------------	---------------------------------

### See Also

[Selection](#), [SelectData](#), [SelectString](#)

## SelectData Method

Selects an item based on its Data and performs an action

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HCM_SELECTDATA, wAction, (LPARAM)lpData);  
HCSelctData(wAction, lpData);
```

#### C++

```
bSuccess = [CHCombObj.]SelectData(lpData, wAction);
```

#### VBX

```
[form.][control.]SelectedData[ = strData]
```

### Arguments/Parameters

WORD wAction	An <u>action code</u>
void FAR *lpData	A pointer to the Data to match
strData	A string representing the Data to match

### Return Value

BOOL bSuccess	TRUE if selection is successful
---------------	---------------------------------

### See Also

SelectCode, Selection, SelectString

## SelectString Method

Selects an item based on some or all of its Text

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HCM_SELECTSTRING, (WPARAM)iStart,  
(LPARAM)lpText);
```

#### C++

```
bSuccess = [CHCombObj.]SelectString(iStart, lpText);
```

#### VBX

```
bSuccess = VCSelectString(control.hWnd, iStart, lpText);
```

### Arguments/Parameters

<code>int iStart</code>	The index at which to start searching
<code>LPSTR lpText</code>	Complete or partial Text of the item to select

### Return Value

<code>BOOL bSuccess</code>	TRUE if selection is successful
----------------------------	---------------------------------

### See Also

[SelectCode](#), [SelectData](#), [Selection](#)

## HComb Events

DoubleClick

SelectChange

SpaceError

KillFocus

SetFocus

### DoubleClick Event

Occurs whenever the user has double-clicks on an item using the mouse

#### Usage

##### C/C++

Notification code: HCN\_DBLCLICK

##### VBX

Sub *ctlname\_DblClick*

#### Return Value

Not Used

## KillFocus Event

Occurs whenever the control loses input focus

### Usage

#### C/C++

Notification code: **HCN\_KILLFOCUS**

#### VBX

**Sub** *ctlname\_KillFocus*

### Return Value

Not Used

## SelectChange Event

Occurs whenever the Selection has changed

### Usage

#### C/C++

Notification code: **HCN\_SELCHANGE**

#### VBX

**Sub** *ctlname\_Change*

### Return Value

Not Used

## SetFocus Event

Occurs whenever the control gains input focus

### Usage

#### C/C++

Notification code: **HCN\_SETFOCUS**

#### VBX

**Sub** *ctlname\_SetFocus*

### Return Value

Not Used

## SpaceError Event

The control is unable to perform an operation because of memory constraints

### Usage

#### C/C++

Notification code: **HCN\_ERRSPACE**

#### VBX

**Sub** *ctlname*\_ErrSpace

### Return Value

Not Used





## HComb Window Text

This sample window text string is expanded below to show the meaning of each component:

`%132%ms%ni###-####`

- `%` Required placeholder.
- `132` The DropHeight in pixels.
- `%` Required placeholder.
- `ms` The DataClass and DataType indicators -- in this case, HC\_MASK and HT\_STRING.
- `%` Optional placeholder. Use if the list contains Codes.
- `ni` The CodeClass and CodeType indicators -- in this case, HC\_NUMBER and HT\_INTEGER. Optional. Use if the list contains Codes
- `###-####` The Format string -- in this case, a seven-digit telephone number.

## HEdit, The WinWidgets Edit Control

- ▣ Attributes
- ▣ Methods
- ▣ Events

HEdit is a two-mode edit control, designed to display, edit and validate all of the data types supported by the [DataEngine](#). Upon receiving focus, the control enters Edit mode. Edit mode employs standard display formats derived from the [Windows International Settings](#), and provides smart templates for editing. Upon losing focus, the control parses its [Text](#) to update its [Data](#) and switches to Display mode. Display mode is non-interactive and provides extensive formatting options for dates, times and numbers. The user may also toggle between Edit and Display modes with the preview key, F2.

To send or retrieve data from the HEdit control, use the [Data](#) or [Text](#) property. The control can be Hot-Linked to a variable within your program, allowing the control to update the variable automatically whenever it parses its Text.

HEdit has standard and 3D [border styles](#) and the ability to [highlight](#) itself upon gaining focus, helping users track their position on forms. The keyboard interface is identical to the standard Windows edit control, supporting both the new commands (Ctl-X, Ctl-C, Ctl-V and Ctl-Z) and old commands (Shift-Del, Ctl-Ins, Shift-Ins and Ctl-Del) for cut, copy, paste and clear. HEdit also provides an overwrite mode for String data types that can be toggled on and off with the Insert key.

### Additional Topics

- ▣ Hot-Linking HEdit to your data
- ▣ Data-Awareness in VB 3.0
- ▣ Data Validation

## Data Validation with HEdit

The HEdit control provides the ability to validate user input using a validation callback procedure. The control will call this procedure with the new Data it has parsed from its Text. An application may modify the data to restrict it to a particular range, or provide the user with a warning message or beep. By returning TRUE from the procedure, the application may force the focus to return to the HEdit control, though this is not a recommended application design.

A validation procedure should be declared as follows:

```
BOOL FAR PASCAL __export MyProc(HWND hwnd, UINT id, LPVOID lpData);
```

When the procedure is called, *hwnd* contains the handle of the HEdit control, *id* is the control's ID and *lpData* is a pointer to the parsed Data, of the DataClass and DataType that were specified for the control. The procedure should return FALSE unless the control is to retain the input focus.

For more information about installing a validation procedure see the ValidateProc attribute. To force the control to validate, use the Validate method.

## HEdit Attributes

<u>AutoHScroll</u>	<u>DisplayAlign</u>	<u>Password</u>
<u>Background</u>	<u>EditAlign</u>	<u>PasswordChar</u>
<u>BorderStyle</u>	<u>Font</u>	<u>Quiet</u>
<u>Case</u>	<u>Format</u>	<u>ScrollPos</u>
<u>Changed</u>	<u>HiliteBrush</u>	<u>Selection</u>
<u>Data</u>	<u>HiliteOnFocus</u>	<u>State</u>
<u>DataClass</u>	<u>Hunger</u>	<u>Text</u>
<u>DataLink</u>	<u>MaxTextLen</u>	<u>TextColor</u>
<u>DataSize</u>	<u>NoHideSel</u>	<u>TextLength</u>
<u>DataType</u>	<u>OverwriteMode</u>	<u>ValidateProc</u>

## AutoHScroll Attribute

When set, the Text automatically scrolls horizontally when the caret nears either end of the window.

### Usage

#### C/C++

Window Style: HES\_AUTOHSCROLL

#### VBX

[*form.*] [*control.*] **AutoHScroll**

## Background Attribute

The color or pattern used to paint the background of the edit control

### Usage

#### C

```
hbrBkgnd = (HBRUSH)SendMessage(hWnd, HEM_GETBKGNDBRUSH, 0, 0L);  
hbrOldBkgnd = (HBRUSH)SendMessage(hWnd, HEM_SETBKGNDBRUSH,  
(WPARAM)hNewBrush, 0L);
```

#### C++

##### OWL

```
hbrBkgnd = [THEditObj.]GetBkgndBrush(void);  
hbrOldBkgnd = [THEditObj.]SetBkgndBrush(hbrNewBrush);
```

##### MFC

```
pBkgnd = [CEditObj.]GetBkgndBrush(void);  
pOldBkgnd = [CEditObj.]SetBkgndBrush(pNewBrush);
```

#### VBX

```
[form.][control.]BackColor[ = color ]
```

See *Visual Basic Language Reference*, "BackColor, ForeColor Properties"

### Arguments/Parameters

HBRUSH hbrNewBrush	Handle of a new brush to be set as the background brush
CBrush pNewBrush	Pointer to a CBrush object containing the new background brush handle

### Return values

HBRUSH hbrBkgnd	Handle to the current background brush
HBRUSH hbrOldBkgnd	Handle to the previous background brush
CBrush *pBkgnd	Pointer to a CBrush object containing the current background brush handle
CBrush *pOldBkgnd	Pointer to a CBrush object containing the previous background brush handle

### Remarks

C and C++ applications are responsible for destroying any brushes they create.

## BorderStyle Attribute

HEdit supports four different border styles: none, standard, indented and bump.

### Usage

#### C/C++

Window Styles:

HES\_BORDER3D

HES\_EXTRUDE

#### VBX

[*form.*][*control.*]**BorderStyle**[= *None/Standard/Indented/Bump*]

### Remarks

The BorderStyle attribute is read-only at run time.

## Case Attribute

When set, Text can be converted to all upper-case or all lower-case.

### Usage

#### C/C++

Window Styles:

HES\_LOWERCASE

HES\_UPPERCASE

#### VBX

[*form.*][*control.*]**Case**

### Remarks

The Case attribute is read-only at run time.

## Changed Attribute

A Boolean value indicating if the Data has been changed since it was last set

### Usage

#### C

```
bChanged = (BOOL)SendMessage(hWnd, HEM_HASCHANGED, 0, 0L);  
SendMessage(hWnd, HEM_SETCANGED, bVal, 0L);
```

#### C++

```
bChanged = [CHEditObj.]HasChanged();  
[CHEditObj.]SetChanged(bVal);
```

#### VBX

Not Used

### Arguments/Parameters

BOOL bVal

The new value for the Changed attribute

### Return Value

BOOL bChanged

TRUE if the Data has been changed since it was last set

### See Also

[Change event](#)



## Data Attribute

The native (binary) data displayed by the control, of a type defined in the [Data Engine](#) chapter of this manual

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HEM_GETDATA, (WPARAM) iMaxBytes,  
(LPARAM) lpData);  
lBytesCopied = SendMessage(hWnd, HEM_SETDATA, 0, (LPARAM) lpData);
```

#### C++

```
LONG lBytesCopied = [CHEditObj.]GetData(lpData [, iMaxBytes = 0] );  
LONG lBytesCopied = [CHEditObj.]SetData(lpData);
```

#### VBX

```
[form.][control.]Data [= string]
```

### Arguments/Parameters

void FAR *lpData	Pointer to data
int iMaxBytes	Maximum number of bytes to copy (used only for strings)

### Return values

LONG lBytesCopied	Number of bytes actually copied to or from the control
-------------------	--

### Remarks

In Visual Basic, both the Data and Text properties are strings. The Data property is generally in a 'bare' format, while the Text property is formatted according to the Format string. For instance, a date control may have a Data property of "12/15/93" and a Text property of "Dec 15, 1993." This difference also applies to the Mask, Currency and Number DataClasses. For Masks, the Data property strips literals.

### See Also

[DataClassHEdit\\_Attr\\_DataClass](#), [DataLinkHEdit\\_Attr\\_DataLink](#), [DataTypeHEdit\\_Attr\\_DataType](#), [Text](#)

## DataClass Attribute

One of the data classes defined in the [Data Engine](#) chapter.

### Usage

#### C

```
cDataClass = (char)SendMessage(hWnd, HEM_GETDATACLASS, 0, 0L);
```

#### C++

```
cDataClass = [CHEditObj.]GetDataClass();
```

#### VBX

```
[form.][control.]DataClass
```

### Return values

char cDataClass                                  One of the [data class character codes](#)

### Remarks

The DataClass attribute is read-only at run time. Control over this attribute is provided by the control design dialogs in Dialog Editor and Resource Workshop, and by the Properties dialog in Visual Basic. When creating windows dynamically in C and C++, the DataClass is included in the [WindowText](#).

### See Also

[Data](#), [DataType](#), [Text](#)

## DataLink Attribute

A pointer to the variable or buffer that will be updated when the Data is changed

### Usage

#### C

```
lpLink = (void FAR *)SendMessage(hWnd, HEM_GETDATALINK, 0, 0L);  
lBytesCopied = (LONG)SendMessage(hWnd, HEM_SETDATALINK,  
(WPARAM)bSetData, (LPARAM)(LPVOID) lpBuf);
```

#### C++

```
lpLink = [CHEditObj.]GetDataLink();  
lBytesCopied = [CHEditObj.]SetDataLink(lpBuf[, bSetData = TRUE]);
```

#### VBX

Not Used

### Arguments/Parameters

void FAR *lpBuf	Pointer to program data
BOOL bSetData	If TRUE, the control initializes the <u>Data</u> to the contents of lpBuf. If FALSE lpBuf is updated to the current Data.

### Return values

void FAR *lpLink	Pointer to the current data link. NULL if no DataLink has been set.
LONG lBytesCopied	Contains the number of bytes copied from lpBuf

### Remarks

Data in the edit control will be set to the contents of lpBuf.

### See Also

Data, Data Types, Update

## DataSize Attribute

The size of the Data in bytes

### Usage

#### C

```
iSize = (int)SendMessage(hWnd, HEM_GETDATASIZE, 0, 0L);
```

#### C++

```
iSize = [CHEditObj.]GetDataSize();
```

#### VBX

```
[form.][control.]DataSize
```

### Return values

int iSize

The size of the data in bytes.

### Remarks

The DataSize property in Visual Basic is read-only and run-time-only.

## DataType Attribute

One of the data types defined in the [DataEngine](#) chapter.

### Usage

#### C

```
cDataType = (char)SendMessage(hWnd, HEM_GETDATATYPE, 0, 0L);
```

#### C++

```
cDataType = [CEditObj.]GetDataType();
```

#### VBX

```
[form.][control.]DataType
```

### Return values

char cDataType

One of the [data type character codes](#)

### Remarks

The DataType attribute is read-only at run time. Control over this attribute is provided by the control design dialogs in Dialog Editor and Resource Workshop, and by the Properties dialog in Visual Basic. When creating windows dynamically in C and C++, the DataType is included in the [WindowText](#).

### See Also

[Data](#), [DataClass](#)

## Display-Mode Alignment Attribute

The justification of the Text in display mode

### Usage

#### C/C++

Window Styles:

HES\_DISPLAYLEFT

HES\_DISPLAYCENTER

HES\_DISPLAYRIGHT

#### VBX

[*form.*] [*control.*] **AlignDisplay**

### Remarks

The Display-Mode Alignment attribute is read-only at run time.

### See Also

Edit-Mode Alignment

## Edit-Mode Alignment Attribute

The justification of the Text in display mode

### Usage

#### C/C++

Window Styles:

HES\_EDITLEFT

HES\_EDITRIGHT

#### VBX

[*form.*][*control.*]**AlignEdit**

### Remarks

The Edit-Mode Alignment attribute is read-only at run time.

### See Also

Display-Mode Alignment

## Font Attribute

The font used by the control

### Usage

#### C

```
hfFont = (HFONT)SendMessage(hWnd, HEM_GETFONT, 0, 0L);  
hfOldFont = SendMessage(hWnd, HEM_SETFONT, (WPARAM)hfNewFont,  
(LPARAM)bRedraw);
```

#### C++

##### OWL

```
hfFont = [THEditObj].GetFont();  
hfOldFont[THEditObj].SetFont(hfNewFont [, bRedraw = TRUE] );
```

##### MFC

```
pFont = [CEditObj].GetFont();  
pOldFont[CEditObj].SetFont(pNewFont [, bRedraw = TRUE] );
```

#### VBX

```
[form].[control].FontBold[= Boolean]  
[form].[control].FontItalic[= Boolean]  
[form].[control].FontName[= font]  
[form].[control].FontSize[= points]  
[form].[control].FontStrikethru[= Boolean]  
[form].[control].FontUnderline[= Boolean]
```

See Visual Basic Language Reference, "FontName Property"

### Arguments/Parameters

HFONT hfNewFont	Handle of the font to be set
CFont *pFont	Pointer to a CFont object containing the handle to the font to be set
BOOL bRedraw	A value of TRUE causes the control to repaint immediately

### Return values

HFONT hfFont	Handle to the control's current font
HFONT hfOldFont	Handle to the control's previous font
CFont *hfFont	Pointer to a CFont object containing the handle to the control's current font
CFont *fOldFont	Pointer to a CFont object containing the handle to the control's previous font

### Remarks

C and C++ applications are responsible for destroying any fonts they create.





## Format Attribute

A NULL-terminated character string that describes the way the Data is to be displayed

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HEM_GETFORMAT, (WPARAM)iMaxBytes,  
(LPARAM)lpstrBuf);  
lBytesCopied = SendMessage(hWnd, HEM_GETFORMAT, (WPARAM)bRedraw,  
(LPARAM)lpstrBuf);
```

#### C++

```
lBytesCopied = [CHEditObj.]GetFormat(lpstrBuf [, iMaxBytes = -1] );  
lBytesCopied = [CHEditObj.]SetFormat(lpstrBuf [, bRedraw = TRUE] );
```

#### VBX

```
[form.][control.]Format[= string]
```

### Arguments/Parameters

LPSTR lpstrBuf	Buffer that contains a new format string or will receive the existing one
int iMaxBytes	The maximum bytes to copy to the buffer. A value of -1 copies the entire format string.
BOOL bRedraw	A value of TRUE causes the control to redraw immediately.

### Return values

LONG lBytesCopied	The number of bytes actually copied to or from the buffer
-------------------	---

### Remarks

An initial format string is contained in the WindowText when a control is created.

### See Also

Text

## HiliteOnFocus Attribute

When set, the HiliteBrush is used to paint the background when the control receives input focus. If no HiliteBrush is selected, the control uses a white brush.

### Usage

#### C/C++

Window Style: HES\_HILITE

#### VBX

[*form.*] [*control.*] **HiliteOnFocus**

### Remarks

The HiliteOnFocus attribute is read-only at run time.

## HiliteBrush Attribute

The color or pattern used to paint the background when the control receives focus and the HiliteOnFocus property is TRUE

### Usage

#### C

```
hbrHilite = SendMessage(hWnd, HEM_GETHILITEBRUSH, 0, 0L);  
hbrOldHilite = SendMessage(hWnd, HEM_SETHILITEBRUSH,  
(WPARAM)hbrNewHilite, 0L);
```

#### C++

##### OWL

```
hbrHilite = [THEditObj.]GetHiliteBrush();  
hbrOldHilite = [THEditObj.]SetHiliteBrush(hbrNewHilite);
```

##### MFC

```
pHilite = [CEditObj.]GetHiliteBrush();  
pOldHilite = [CEditObj.]SetHiliteBrush(pNewHilite);
```

#### VBX

```
[form.][control.]HiliteColor[= color]
```

### Arguments/Parameters

HBRUSH hbrNewHilite	Handle of the new brush
CBrush *pNewHilite	Pointer to a CBrush object containing the handle of the new brush

### Return values

HBRUSH hbrHilite	Handle of the control's current brush
HBRUSH hbrOldHilite	Handle of the control's previous brush
CBrush *pHilite	Pointer to a CBrush object containing the handle of the current brush
CBrush *pOldHilite	Pointer to a CBrush object containing the handle of the previous brush

### Remarks

C and C++ applications are responsible for destroying any brushes they create.

## Hunger Attribute

When set, the control swallows *Enter* and *Esc* keyboard messages and notifies its parent.

### Usage

#### C/C++

Window Style: HES\_HUNGER

#### VBX

[*form.*] [*control.*] **Hunger**

### Remarks

The Hunger attribute is obsolete and is included here for backward compatibility. We recommend that C and C++ programmers use a Filter Procedure or Dynamic Subclassing, respectively, to implement this functionality. Hunger is read-only at run time.

## MaxTextLen Attribute

The maximum number of characters that can be entered into the edit box with the `HC_STRING DataClass`

### Usage

#### C

```
iMaxLen = (int)SendMessage(hWnd, HEM_GETMAXTEXTLEN, 0, 0L);  
SendMessage(hWnd, HEM_SETMAXTEXTLEN, (WPARAM)iLen, 0L);
```

#### C++

```
iMaxLen = [CHEditObj.]GetMaxTextLen();  
[CHEditObj.]SetMaxTextLen(iLen);
```

#### VBX

```
[form.][control.]MaxTextLen[= iLen]
```

### Arguments/Parameters

<code>int iLen</code>	New maximum length. A value of -1 removes the maximum text length
-----------------------	---

### Return values

<code>int iMaxLen</code>	The currently set maximum text length.
--------------------------	--

### Remarks

The `MaxTextLen` can only be set for the `HC_STRING DataClass`. Maximum lengths for other classes are determined by their Format strings.

## NoHideSel Attribute

When set, the control hides the Selection on losing input focus and redisplay the previous Selection on regaining focus.

### Usage

#### C/C++

Window Style: HES\_NOHIDSESEL

#### VBX

[*form.*] [*control.*]NoHideSel

### Remarks

NoHideSel is read-only at run time.

## OverwriteMode Attribute

Determines whether text overwrites existing text or is inserted as it is input into a control of the HC\_STRING DataClass.

### Usage

#### C

```
bOverwrite = (BOOL)SendMessage(hWnd, HEM_GETOVERWRITEMODE, 0, 0L);  
SendMessage(hWnd, HEM_SETOVERWRITEMODE, (WPARAM)bMode, 0L);
```

#### C++

```
bOverwrite = [CHEditObj.]GetOverwriteMode();  
[CHEditObj.]SetOverwriteMode( [bMode = TRUE] );
```

#### VBX

```
[form.][control.]OverwriteMode[= bMode]
```

### Arguments/Parameters

BOOL bMode

TRUE for overwrite, FALSE for insert

### Return values

BOOL bOverwrite

Current mode of the control, TRUE for overwrite, FALSE for insert

### Remarks

Overwrite mode is only used with the HC\_STRING DataClass. Other DataClasses insert or overwrite characters based on the position of the caret in the editing template.



## Password Attribute

When set, all characters input are displayed as the character specified by the PasswordChar attribute.

### Usage

#### C/C++

Window Style: HES\_PASSWORD

#### VBX

[*form.*] [*control.*] Password

### Remarks

Password is read-only at run time.

## PasswordChar Attribute

A character which will appear in the control's display when any character is input if the control's Password attribute is also set.

### Usage

#### C

```
cPwdChar = (char)SendMessage(hWnd, HEM_GETPASSWORDCHAR, 0, 0L);  
SendMessage(hWnd, HEM_SETPASSWORDCHAR, (WPARAM)cChar, 0L);
```

#### C++

```
cPwdChar = [CHEditObj.]GetPasswordChar();  
[CHEditObj.]SetPasswordChar(cChar);
```

#### VBX

```
[form.][control.]PasswordChar[= cChar]
```

### Arguments/Parameters

char cChar

The character that appear in the display

### Return values

char cPwdChar

The currently set PasswordChar

## Quiet Attribute

When the control is in Quiet mode, it does not send notification messages to its parent. VBX controls will not fire events in Quiet mode.

### Usage

#### C

```
bIsQuiet = SendMessage(hWnd, HEM_ISQUIET, 0, 0L);  
SendMessage(hWnd, HEM_BEQUIET, bValue, 0L);
```

#### C++

```
bIsQuiet = [CHEditObj.]IsQuiet  
[CHEditObj.]BeQuiet(bValue);
```

#### VBX

```
bIsQuiet = SendMessage(control.hWnd, HEM_ISQUIET, 0, 0L);  
SendMessage(control.hWnd, HEM_BEQUIET, bValue, 0L)
```

See [VBX Advanced Topics](#)

### Arguments/Parameters

BOOL bValue

TRUE turns on Quiet mode, FALSE turns it off

### Return values

BOOL bIsQuiet

TRUE if the control is in Quiet mode

## ScrollPos Attribute

The number of characters that have been scrolled out of the control's client area

### Usage

#### C

```
iScrollPos = (int)SendMessage(hWnd, HEM_GETSCROLLPOS, 0, 0L);  
iScrollPos = SendMessage(hWnd, HEM_SETSCROLLPOS, (WPARAM)iScroll,  
(LPARAM)bRedraw);
```

#### C++

```
iScrollPos = [CEditObj.]GetScrollPos();  
iScrollPos = [CEditObj.]SetScrollPos(iScroll [, bRedraw = TRUE]);
```

#### VBX

```
[form.][control.]ScrollPos[= iScroll]
```

### Arguments/Parameters

int iScroll

The number of characters to scroll off the left side for left justified text and off the right side for right justified text

BOOL bRedraw

A value of TRUE causes the control to repaint itself immediately

### Return values

int iScrollPos

The number of characters scrolled off the left side for left justified text and off the right side for right justified text

## Selection Attribute

The text within the control that is currently selected (hilited)

### Usage

#### C

```
lSel = SendMessage(hWnd, HEM_GETSEL, 0, 0L);  
SendMessage(hWnd, HEM_GETSELTEXT, (WPARAM) iMaxBytes,  
    (LPARAM) lpstrSelText);  
SendMessage(hWnd, HEM_SETSEL, 0, (LPARAM) lNewSel);  
SendMessage(hWnd, HEM_REPLACESEL, 0, (LPARAM) lpStr);
```

#### C++

```
lSel = [CHEditObj.]GetSel();  
[CHEditObj.]GetSelText(lpstrSelText[, iMaxBytes = -1]);  
[CHEditObj.]SetSel(lNewSel);  
[CHEditObj.]ReplaceSel(lpStr);
```

#### VBX

```
[form.][control.]SelLength[= length]  
[form.][control.]SelStart[= index]  
[form.][control.]SelText[= stringexpression]  
See Visual Basic Language Reference, "SelLength, SelStart, SelText Properties"
```

### Arguments/Parameters

LPSTR lpstrSelText	Buffer to receive selected text
int iMaxBytes	Maximum number of characters to copy into lpstrSelText
LONG lNewSel	Contains the starting position in the low-order word and the character position of the first non-selected character after the selection in the high-order word
LPSTR lpStr	String to replace selected Text

### Return values

LONG lSel	Contains the starting position in the low-order word and the character position of the first non-selected character after the selection in the high-order word
-----------	--

### Remarks

Visual Basic programmers can replace the Selection by assigning a new value to the **SelText** property.

## State Attribute

A collection of flags describing the state of the control

### Usage

#### C

```
lState = SendMessage(hWnd, HEM_GETSTATE, 0, 0L);
```

#### C++

```
lState = [CHEditObj.]GetState();
```

#### VBX

Not Used

### Return values

LONG lState

A long value representing the control's state

### Remarks

The State property is obsolete. It is included here for backwards compatibility

### See Also

Changed

## HEdit State Flags

Constant	Value	Meaning
HEF_DISPLAYMODE	0x0002	Control is in Display mode
HEF_RIGHT	0x0004	Text is right justified in current mode
HEF_CENTER	0x0008	Text is centered in the current mode
HEF_CHANGED	0x0080	Data has changed since last HEM_SETDATA

## Text Attribute

A character string representing formatted data. When the Text is set, the control will first parse the text to set its Data, then format the Data according to the Format.

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HEM_GETTEXT, (WPARAM) iMaxBytes,  
    (LPARAM) lpBuf);  
bResult = SendMessage(hWnd, HEM_SETTEXT, 0, (LPARAM) lpBuf);
```

#### C++

```
lBytesCopied = [CHEditObj].GetText(lpBuf [, iMaxBytes = -1]);  
bResult = [CHEditObj].SetText(lpBuf);
```

#### VBX

```
[form].[control].Text[= stringexpression]  
See Visual Basic Language Reference, "Text Property"
```

### Arguments/Parameters

LPSTR lpBuf	A buffer for the control Text
int iMaxBytes	The maximum number of bytes to copy to lpBuf

### Return values

LONG lBytesCopied	The number of bytes actually copied to lpBuf
BOOL bResult	TRUE if Text has been set successfully

### Remarks

The Text of the control should not be confused with the caption or "window text". The edit control's window text contains DataType and DataClass information and the Format string. The window text is only used at the time of creation and is not updated if this information changes. The standard windows messages, WM\_GETTEXT and WM\_SETTEXT are processed by HEdit to return and set the Text, not the window text.

### See Also

Format, TextColor, TextLength



## TextColor Attribute

A color used to paint the Text

### Usage

#### C

```
crTextColor = SendMessage(hWnd, HEM_GETTEXTCOLOR, (WPARAM)bNeg, 0L);  
SendMessage(hWnd, HEM_SETTEXTCOLOR, (WPARAM)bNeg, crColor);
```

#### C++

```
crTextColor = [CHEditObj.]GetTextColor([bNeg = FALSE]);  
[CHEditObj.]GetTextColor(crColor [, bNeg = FALSE]);
```

#### VBX

```
[form.][control.]TextColor[= color]  
[form.][control.]TextColor_Neg[= color]
```

### Arguments/Parameters

BOOL bNeg

TRUE specifies the TextColor for negative numbers, which can be set independently of the TextColor for positive numbers

COLORREF crColor

Specifies the new color

### Return values

COLORREF crTextColor

Contains the TextColor specified by bNeg

### See Also

[Format](#), [Text](#)

## TextLength Attribute

The length of Text in characters, excluding the terminating NULL character

### Usage

#### C

```
lTextLen = SendMessage(hWnd, HEM_GETTEXTLEN, 0, 0L);
```

#### C++

```
lTextLen = [CHEditObj.]GetTextLen();
```

#### VBX

```
Len([control.]Text)
```

### Return values

LONG lTextLen

The length of the Text in characters

### See Also

[Text](#)

## ValidateProc Attribute

A validation callback procedure that can be installed to validate user-input. The procedure is called whenever the control parses its Text into Data, normally on losing focus. It is also called in response to the Validate method.

### Usage

#### C

```
lpfnValProc = SendMessage(hWnd, HEM_GETVALIDATE, 0, 0L);  
SendMessage(hWnd, HEM_SETVALIDATE, 0, (LPARAM)lpfnNewValProc);
```

#### C++

```
pfnValProc = [CHEditObj].GetValidate();  
[CHEditObj].SetValidate(lpfnNewValProc);
```

#### VBX

Not Used

### Arguments/Parameters

FARPROC lpfnNewValProc

A pointer to the validation procedure to install, or NULL to uninstall a validation procedure.

### Return values

FARPROC lpfnValProc

A pointer to the currently installed validation procedure

### Remarks

C++ applications requiring data validation should prepare a Validation Procedure in C or as a static member of a CHEdit-derived class. Ordinary member functions cannot be used as callback procedures.

### Examples

### See Also

[Using a Data Validation Procedure](#)

## HEdit Methods

<a href="#">Clear</a>	<a href="#">Paste</a>	<a href="#">Validate</a>
<a href="#">Copy</a>	<a href="#">Undo</a>	
<a href="#">Cut</a>	<a href="#">Update</a>	

## Clear Method

Removes selected Text

### Usage

#### C

```
SendMessage(hWnd, WM_CLEAR, 0, 0L);
```

#### C++

```
[CHEditObj.]Clear();
```

#### VBX

```
SendMessage(control(hWnd), WM_CLEAR, 0, 0L)
```

See [VBX Advanced Topics](#)

### See Also

[Copy](#), [Cut](#), [Paste](#), [Undo](#)

## Copy Method

Copies selected Text to the clipboard

### Usage

#### C

```
SendMessage(hWnd, WM_COPY, 0, 0L);
```

#### C++

```
[CHEditObj.]Copy();
```

#### VBX

```
SendMessage(control(hWnd), WM_COPY, 0, 0L);
```

See [VBX Advanced Topics](#)

### See Also

[Clear](#), [Cut](#), [Paste](#), [Undo](#)

## Cut Method

Removes selected Text and copies it to the Clipboard

### Usage

#### C

```
SendMessage(hWnd, WM_CUT, 0, 0L);
```

#### C++

```
[CHEditObj.]Cut();
```

#### VBX

```
SendMessage(control(hWnd), WM_CUT, 0, 0L);
```

See [VBX Advanced Topics](#)

### See Also

[Clear](#), [Copy](#), [Paste](#), [Undo](#)

## Paste Method

Copies from the clipboard to the control's current caret position

### Usage

#### C

```
SendMessage(hWnd, WM_PASTE, 0, 0L);
```

#### C++

```
[CHEditObj.]Paste();
```

#### VBX

```
SendMessage(control(hWnd), WM_PASTE, 0, 0L)
```

See [VBX Advanced Topics](#)

### See Also

[C](#), [Copy](#), [Cut](#), [Undo](#)

## Undo Method

Redisplays Text to reflect Data

### Usage

#### C

```
SendMessage(hWnd, EM_UNDO, 0, 0L);
```

#### C++

```
[CHEditObj.]Undo();
```

#### VBX

```
SendMessage(control(hWnd), EM_UNDO, 0, 0L);
```

See [VBX Advanced Topics](#)

### See Also

[C](#), [Copy](#), [Cut](#), [Paste](#)



## Update Method

Causes the control to update its Text and Data

### Usage

#### C

```
SendMessage(hWnd, HEM_UPDATE, (WPARAM)bParseText, (LPARAM)bUpdateText);
```

#### C++

```
[CHEditObj.]Update([bParseText = FALSE [, bUpdateText = TRUE]]);
```

#### VBX

```
HEUpdate(control(hWnd, bParseText, bUpdateText)
```

### Arguments/Parameters

BOOL bParseText

bParseText%

TRUE causes Data to be updated based on current Text.

BOOL bUpdateText

bUpdateText%

TRUE causes Text to be updated to reflect current Data. If bParseText is also TRUE, the Text will be parsed first.

### Remarks

This message is typically sent with bParseText=FALSE, bUpdateText=TRUE when the application has changed the Data and wants the control to reflect the change, and bParseText=TRUE when the control has focus and the application wants the control to parse its Text before the application uses the Data.

### See Also

[Data](#), [Text](#)

## Validate Method

Calls a previously set validation callback procedure

### Usage

#### C

```
iResult = (int)SendMessage(hWnd, HEM_VALIDATE, 0, 0L);
```

#### C++

```
iResult = Validate();
```

#### VBX

Not Used

### Return values

int iResult

The value returned by the validation procedure or -1 if no validation procedure has been installed

### See Also

[Using a Data Validation Procedure](#), [ValidateProc](#) attribute

## HEdit Events

<u>Change</u>	<u>KillFocus</u>	<u>SpaceError</u>
<u>HScroll</u>	<u>MaxText</u>	<u>Update</u>
<u>Invalid</u>	<u>SetFocus</u>	

## Change Event

The control's Text has been altered by user input.

### Usage

#### C/C++

Notification code: **HEN\_CHANGE**

#### VBX

**Sub** *ctlname\_Change* (*Index As Integer*)

### Return Value

Not Used

## HScroll Event

The user clicked the horizontal scroll bar.

### Usage

#### C/C++

Notification code: **HEN\_HSCROLL**

#### VBX

**Sub** *ctlname\_HScroll* (Index **As Integer**)

### Return Value

Not Used

## Invalid Event

The user has entered an invalid date or other data item

### Usage

#### C/C++

Notification code: **HEN\_INVALID**

#### VBX

**Sub** *ctlname\_Invalid* (*Index As Integer*)

### Return Value

Returning TRUE prevents the control from losing focus.

### Remarks

This event only occurs for very general formatting errors (e.g. typing 13/32/93 in a date field). Range checking and other application-specific error checking must be done in a ValidateProc.

## KillFocus Event

The control has lost input focus.

### Usage

#### C/C++

Notification code: **HEN\_KILLFOCUS**

#### VBX

**Sub** *ctlname\_KillFocus* (Index **As Integer**)

### Return Value

Not Used

## MaxText Event

The Text has reached the MaxTextLen.

### Usage

#### C/C++

Notification code: HEN\_MAXTEXT

#### VBX

**Sub** *ctlname\_MaxText* (Index **As Integer**)

### Return Value

Not Used

## SetFocus Event

The control has gained input focus.

### Usage

#### C/C++

Notification code: **HEN\_SETFOCUS**

#### VBX

**Sub** *ctlname\_SetFocus* (Index **As Integer**)

### Return Value

Not Used



## SpaceError Event

The control was unable to allocate memory.

### Usage

#### C/C++

Notification code: **HEN\_ERRSPACE**

#### VBX

**Sub** *ctlname\_ErrSpace* (Index **As Integer**)

### Return Value

Not Used

## Update Event

The control is about to display altered Text.

### Usage

#### C/C++

Notification code: **HEN\_UPDATE**

#### VBX

**Sub** *ctlname\_Update* (Index **As Integer**)

### Return Value

Not Used

## HEdit Window Text

This sample window text string is expanded below to show the meaning of each component:

`%ms###-####`

`%` Required placeholder.

`ms` The DataClass and DataType indicators -- in this case, HC\_MASK and HT\_STRING.

`###-####` The Format string -- in this case, a seven-digit telephone number.

## HGrid, The WinWidgets Grid Control

- ▣ Attributes
- ▣ Methods
- ▣ Events

The HGrid control simplifies the browsing and editing of data tables for both the user and programmer. The HGrid presents data tables in a spreadsheet-like format with each column corresponding to a field in the table and each row to a record. To provide custom display and editing for each field in the table, each column is assigned one of the other WinWidgets, either an Edit control, ListBox, ComboBox or CheckBox. For a more details, see An Introduction to the HGrid, or any of the topics listed below.

### Additional Topics

- ▣ Creating and using HGrid resources
- ▣ HGrid Record Structures
- ▣ Using a Record Buffer
- ▣ Initializing the Grid's child controls
- ▣ Hot-Linking a variable to the current record
- ▣ Data Validation in the Grid
- ▣ Custom cursor/selection behavior
- ▣ Using HGrid with MFC's CView class

## An Introduction to the HGrid

The HGrid control provides a flexible, capable tool for browsing and editing tabular data that reduces the requisite programming to a minimum. The HGrid uses tables like those found in databases. A table can contain multiple fields of information for each of many records. A sample table, which might be used in a personnel system, has a record for each employee that contains the fields "Name," "Position," "Date of Birth," "Date of Hire," and "Salary."

The definition of an HGrid table begins with the list of fields it contains. For each field, the HGrid employs a separate control to display and edit the field's data. In the table described above, the "Position" field might use an [HComb](#) control to allow selection from a pre-defined list of employee positions, while the others would use [HEdit](#) controls.

Once the type of control for a field is determined, the field control is prepared as it would be for use in a standard dialog box. For instance, an HEdit control must know the field's data class, type and format, and its display alignment. The HGrid's dialog editor extensions provide an easy way to prepare the HGrid and its field controls. For more details on this process, see the section [Creating and Using HGrid Resources](#).

The HGrid control supports all of the data types supported by the [HEdit](#), [HList](#) and [HComb](#) controls. It can also be "hot-linked" to a data structure that will be updated with the contents of the current record as the user moves within a table. The control provides notification to its parent whenever the user edits a record, moves to a different record or a different field, or tries to scroll past the first or last record in the table. It also maintains a flag for each record that indicates whether the record has been edited.

The user interface of the HGrid control is extremely flexible and familiar to spreadsheet users. Tables can be displayed in Edit or Browse mode and as overlapped, popup, child or MDI child windows. Rows and columns are resizeable, and the grid lines can be toggled on and off. Editing can be in-place or in the toolbar, spreadsheet style. The user may also copy and paste from an HGrid control to a spreadsheet.

## Creating and Using HGrid Resources

In setting up an HGrid control, more information must be provided than for the other WinWidgets. Each field's definition is comparable to the definition of an HEdit or HList control. As the number of fields increases, the amount of information describing the table quickly exceeds the amount that can be stored in the control's window text (the conventional place to store control descriptions). To store its defining information, the HGrid uses a custom resource file format with the extension .GRS and stores only the name of the HGRID resource in the control's window text. To include an HGrid control in an application, a statement should be included in the application's resource (.RC) file similar to the following:

```
MyGrid RT_HGRID "MyGrid.GRS"
```

When the HGrid control is created, it reads the HGRID resource name from its window text and attempts to load the grid from the application's instance. If the control is unable to find the resource, as will occur in a resource editor (which does not load the application), the control looks for the resource as a .GRS file. An application may also direct the control to look for resources in other instances using the HM\_GETINSTANCE message (see Using Custom Resources).

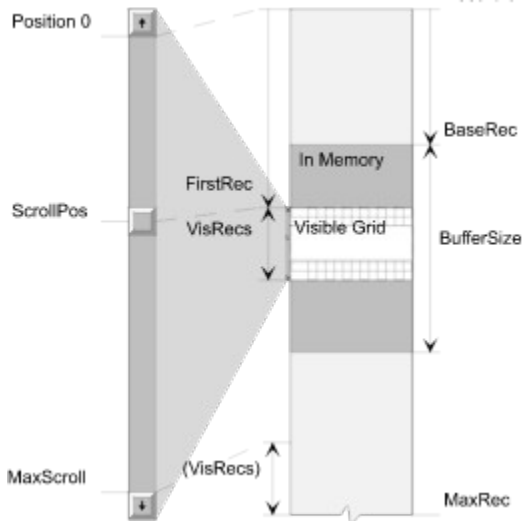
### Using HGrid in a Resource Editor

All preparation of the HGrid control can be accomplished through a resource editor, by following these steps:

1. Open the HGrid Styles dialog box by double-clicking on the control.
2. Enter the control's style options, ID, title and resource name.
3. Open the Field Control Styles dialog box by choosing the **Fields** menu.
4. Use the **Field** menu to browse among fields and the **Edit** menu to insert, delete and copy fields.
5. Press OK to accept any changes to the field information, or Cancel to disregard the changes. The Field Control Styles dialog box will close.
6. Choose **File, Save** and save the HGrid information to a file with the same name entered in the Resource Name edit box.
7. Press OK to accept any changes to the table, or Cancel to disregard the changes. The HGrid Styles dialog box will close.

## Using a Record Buffer

Record buffering allows the HGrid to browse and edit large tables of information without having all of the information in memory. To the user, it appears that the control contains the entire table, but the HGrid is really only requesting and storing data as needed, using the installed BufferProc. The BufferProc is an application-defined callback procedure that the HGrid calls when records are removed from the buffer (by either deletion or displacement caused by newly added records), when new records are needed in the buffer, and when the contents of a record in the buffer have changed.



Once a BufferProc that performs these actions is installed, no other interaction is required to send and retrieve data from the HGrid. There are, however, other aspects of a buffering HGrid control that an application can control to customize or optimize behavior. The picture above shows the attributes of the HGrid control pertinent to record buffering.

BaseRec is a long value that marks the starting position of the record buffer in the table. It is the first record of the table that is currently in memory. BaseRec should *not* be adjusted by an application; use the FirstRec instead.

MaxRec is a long value representing the total number of records in the table. If MaxRec is set by an application the HGrid will provide a virtual scroll range from zero to MaxRec. If MaxRec is not set, it will grow automatically when records are added and/or the BaseRec changes.

BufferSize is an integer value representing the total number of records that the HGrid keeps in memory. The BufferSize can be set by an application in order to optimize performance. However, a minimum BufferSize of twice the number of visible records will always be maintained.

FirstRec is a long value that marks the first record of the table to be displayed in the HGrid. An application can manipulate the viewport by setting the FirstRec. If the FirstRec is set to a position outside the current buffer, the buffer moves automatically by adjusting the BaseRec.

### Notes:

The fact that the HGrid properties FirstRec and MaxRec are both long values allows a buffered HGrid control to handle tables containing millions of records. However, for

practical reasons, the HGrid is limited to 32,767 records at a time (the maximum integer value). In fact, for performance reasons it is wise to maintain a BufferSize of a few hundred or less. The optimal buffer size will depend on a balance of the HGrid's performance and that of the application's data source.

Because the HGrid never contains more than 32,767 records, all record-related attributes of HGrid control, such as the current Anchor and Extent positions, are stored as integer offsets from the BaseRec. This makes it easier to set and retrieve cell positions, which are comprised of two integers rather than an integer for the Field and a long for the Record, but requires the programmer to remember that the record values are not necessarily the absolute positions within the data table.

Another implication of buffering is that selection ranges are always clipped to the current buffer, although this may not be readily apparent to the user.

**See Also**

[Record Buffering in C](#), [Record Buffering in C++](#), [Record Buffering in Visual Basic](#)



## Record Buffering in C

The BufferProc is an application-defined callback procedure, with the following declaration:

```
BUFFERPROC _export MyBufferProc (HWND hwndGrid, WORD wAction, LONG  
lRecNum, LP[MYRECTYPE]lpRecData );
```

The first argument to the BufferProc is the window handle of the grid control, in case the procedure needs to send or retrieve additional information from the HGrid. The second parameter, wAction specifies the action that the BufferProc should take. Possible values for wAction and their meanings are:

- HGB\_RETRIEVE** This message requests the contents of a record from the application. IRecNum is the requested record number. lpRecData points to an empty data buffer that should be filled by the application. The application must return TRUE to add the record to the HGrid.
- HGB\_UPDATE** This message notifies the application that a modified record is being removed from the record buffer. The application should use this opportunity to update the record in its data source.
- HGB\_INSERT** This message informs the application that the user is attempting to insert a new record. IRecNum is the position of the insertion. lpRecData points to an empty data buffer that can be filled by the application if the record needs to be initialized. The application should also use this opportunity to insert the record into its data source. The application must return TRUE to insert the record into the HGrid's record buffer.
- HGB\_DELETE** This message informs the application that the user is attempting to delete a record. IRecNum is the position of the deletion. lpRecData points to a copy of the record's contents. The application should use this opportunity to delete the record from its data source. The application must return TRUE to delete the record from the HGrid's record buffer.
- HGB\_CHANGE** This message notifies the application that a record has been modified. The application can use this opportunity to update the record in its data source, or the application can wait for a HGB\_UPDATE message. If the application processes this message it should return TRUE to reset the record's CHANGED flag.
- HGB\_VALIDATE** This message notifies the application that the user is attempting to leave a changed cell. The application can use this opportunity to validate the cell's new data. IRecNum actually contains an integer column index in the LOWORD and an integer row index in the HIWORD. lpRecData

points to the cell's changed data. If the application processes this message it should return TRUE if the cell's data is *invalid* and the selection will remain at that cell.

## Record Buffering in C++

Since WIDGETS.DLL is written in C, it cannot use member functions of C++ classes with mangled names as callback functions. To provide record buffering capability in our C++ classes we have created a special class, CHGridBuffer (THGridBuffer in OWL), which is subclassed by the application. CHGridBuffer contains a public member function,

```
BOOL AttachBuffer(CHGrid *pGrid);
```

which registers the subclassed buffer object and the CHGrid object to which it is attached with a buffer manager. The buffer manager, in turn, registers a static callback function with the HGrid wrapped by the CHGrid object. When the HGrid calls this callback function, the buffer manager calls one of five virtual functions in the appropriate CHGridBuffer object. These virtual functions must be overridden by the application to perform the storage and retrieval of records to and from the data source.

```
virtual BOOL OnRetrieve(LONG lRecNum, LPVOID lpRecData);  
virtual BOOL OnUpdate(LONG lRecNum, LPVOID lpRecData);  
virtual BOOL OnInsert(LONG lRecNum, LPVOID lpRecData);  
virtual BOOL OnDelete(LONG lRecNum, LPVOID lpRecData);  
virtual BOOL OnChange(LONG lRecNum, LPVOID lpRecData);  
virtual BOOL OnValidate(int iCol, int iRow, LPVOID lpCellData);
```

Their arguments and return values are interpreted as follows:

- OnRetrieve** This function requests the contents of a record from the application. IRecNum is the requested record number. lpRecData points to an empty data buffer that should be filled by the application. The application must return TRUE to add the record to the HGrid.
- OnUpdate** This function notifies the application that a modified record is being removed from the record buffer. The application should use this opportunity to update the record in its data source.
- OnInsert** This function informs the application that the user is attempting to insert a new record. IRecNum is the position of the insertion. lpRecData points to an empty data buffer that can be filled by the application if the record needs to be initialized. The application should also use this opportunity to insert the record into its data source. The application must return TRUE to insert the record into the HGrid's record buffer.
- OnDelete** This function informs the application that the user is attempting to delete a record. IRecNum is the position of the deletion. lpRecData points to a copy of

the record's contents. The application should use this opportunity to delete the record from its data source. The application must return TRUE to delete the record from the HGrid's record buffer.

**OnChange** This function notifies the application that a record has been modified. The application can use this opportunity to update the record in its data source, or can wait for the OnUpdate function to be called. If the application processes this message it should return TRUE to reset the record's CHANGED flag.

**OnValidate** This function notifies the application that the user is attempting to leave a changed cell. The application can use this opportunity to validate the cell's new data. iCol and iRow contain the coordinates of the cell. lpCellData points to the cell's changed data. If the application processes this message it should return TRUE if the cell's data is *invalid* and the selection will remain at that cell.

The CHGridBuffer object is automatically detached from the CHGrid object when the HGrid is destroyed.

## **Record Buffering in Visual Basic**

Users of Visual Basic 3.0 may take advantage of the VBX Grid's Data Awareness features, which include automatic record buffering.

## HGrid Record Structures

The HGrid stores data in a list of records, each of which is a contiguous block of memory. Several of the control's methods, such as [AddRec](#), expect the application to provide or receive a record's data in this format. Often, it is possible to define a structured type using *typedef* that matches the HGrid's record structure exactly, making it very easy to move information back and forth from the control.

The structure of the record block depends on the fields that are defined for the table. Each field is allotted a fixed-length section of the record block at a fixed offset from the block start.

The length of a field's section is determined by the type of data contained in the field. For instance, a double is allocated eight bytes and a short int is allocated two. String types are allocated the number of bytes specified by the *iSize* parameter of the [FieldCreate\(\)](#) method, or the String Length entry of the Field Control Styles dialog. Note that both of these values must include space for the NULL terminator.

The offset of a field's section is simply the sum of the sizes of the fields preceding it in the record. It is important to remember that the HGrid expects and allows no padding between fields. Therefore, it is necessary to compile record structures with single byte alignment. If that is not possible, simply make sure that there are no odd-byte sized fields in the record structure.

### Example

A table contains three fields: a string, a double and [long date](#). The string has a maximum length of 20 characters, including a NULL terminator. The application declares a structure to match the Grid's record structure as follows:

```
typedef struct
{
    char    Name[20];
    double  Amount;
    LONGDATE Date;
}
MYREC,
FAR * LPMYREC;
```

Note that the structure allocates the *maximum* amount of space that the Name field may use. When the Grid copies information to or from a record structure it uses a single `memcpy()` of the entire record. To initialize a single cell's data without employing a record structure, see the [CellData](#) and [CellText](#) attributes.

To add records to the Grid using the [AddRec](#) method, declare a variable of type MYREC, initialize it, then pass a pointer to the structure to the Grid through the `AddRec` method:

```
{
MYREC MyRec;                               // Declare

memset (MyRec, 0, sizeof(MYREC));          // Initialize
strcpy (MyRec.Name, "Dr. Dada");
MyRec.Amount = 4550.32;
MyRec.Date = 19921001;
```

```
                                // Send to Grid
SendMessage (hwndGrid, HGM_ADDREC, 0, (DWORD)MyRec);
}
```

Non-initialized (zero-filled) records can be added by passing NULL in place of the record structure pointer.

## Initializing the Grid's Child Controls

The HGrid control employs the other WinWidgets to display and edit each cell's data. A separate child control is created for each field in the Grid, and is then called upon whenever a cell within that field needs to be displayed or edited.

When it is necessary for an application to deal with the child controls directly, such as the initialization of a ComboBox field, a handle to the child control can be accessed through the FldWindow property. The application may then set the attributes and use the methods defined for that type of control. However, certain attributes should not be used, such as the ValidateProc attribute of the edit control, because they interfere with the Grid's use of controls.

To initialize a ListBox or ComboBox within the Grid, first get a handle to the FldWindow, then use the Add and Insert methods to fill the list. Do not adjust the control's Selection; the Grid sets the Selection based on the each cell's data (see the CellData attribute).

## HGrid Data Validation

The HGrid control provides two means of validating user input. The first and preferable method is through the HGB\_VALIDATE action code sent to a C buffer procedure, or through the OnValidate buffer method in C++. This method provides the row and column indices of the altered cell as well as a pointer to the new CellData. The data can easily be modified or trigger a warning message, etc. Returning TRUE prevents the Grid from altering the cell's data.

The second method is to use the RecChanged event, which occurs **after** after the cell's data is changed. When a RecChanged event occurs, the Marker is set to the row and column of the altered cell. If an application chooses to reset the selection to the altered cell in response to invalid data, it must post an HGM\_SETSELEXTENT message to the control with the coordinates retrieved from the Marker.



## HGrid Attributes

### Grid Attributes

<a href="#"><u>AutoExtend</u></a>	<a href="#"><u>DisableNoScroll</u></a>	<a href="#"><u>Marker</u></a>	<a href="#"><u>Selection</u></a>
<a href="#"><u>Background</u></a>	<a href="#"><u>DragCols</u></a>	<a href="#"><u>MaxRec</u></a>	<a href="#"><u>SingleSelect</u></a>
<a href="#"><u>BaseRec</u></a>	<a href="#"><u>EditInPlace</u></a>	<a href="#"><u>MDIChild</u></a>	<a href="#"><u>State</u></a>
<a href="#"><u>Browse</u></a>	<a href="#"><u>FirstCol</u></a>	<a href="#"><u>NoHideSel</u></a>	<a href="#"><u>Style</u></a>
<a href="#"><u>BtnHeight</u></a>	<a href="#"><u>FirstRec</u></a>	<a href="#"><u>NoLines</u></a>	<a href="#"><u>Title</u></a>
<a href="#"><u>BtnWidth</u></a>	<a href="#"><u>Font</u></a>	<a href="#"><u>Quiet</u></a>	<a href="#"><u>VScroll</u></a>
<a href="#"><u>BufferProc</u></a>	<a href="#"><u>FrozenCols</u></a>	<a href="#"><u>ResizeCols</u></a>	<a href="#"><u>VScrollPos</u></a>
<a href="#"><u>BufferSize</u></a>	<a href="#"><u>HScroll</u></a>	<a href="#"><u>ResizeRows</u></a>	<a href="#"><u>WholeRows</u></a>
<a href="#"><u>ColButtons</u></a>	<a href="#"><u>HScrollPos</u></a>	<a href="#"><u>RowBtns</u></a>	
<a href="#"><u>ColCount</u></a>	<a href="#"><u>KbdDelIns</u></a>	<a href="#"><u>RowCount</u></a>	
<a href="#"><u>ColMap</u></a>	<a href="#"><u>LeaveOnTab</u></a>	<a href="#"><u>RowHeight</u></a>	

### Field Attributes

<a href="#"><u>FldBrowse</u></a>	<a href="#"><u>FldCtlStyle</u></a>	<a href="#"><u>FldFormat</u></a>	<a href="#"><u>FldOffset</u></a>
<a href="#"><u>FldCodeClass</u></a>	<a href="#"><u>FldDataClass</u></a>	<a href="#"><u>FldFormatLen</u></a>	<a href="#"><u>FldWindow</u></a>
<a href="#"><u>FldCodeType</u></a>	<a href="#"><u>FldDataSize</u></a>	<a href="#"><u>FldHidden</u></a>	
<a href="#"><u>FldColWidth</u></a>	<a href="#"><u>FldDataType</u></a>	<a href="#"><u>FldName</u></a>	
<a href="#"><u>FldCtlType</u></a>	<a href="#"><u>FldDropHeight</u></a>	<a href="#"><u>FldNameLen</u></a>	

### Record Attributes

<a href="#"><u>RecBrowse</u></a>	<a href="#"><u>RecData</u></a>	<a href="#"><u>RecNew</u></a>
<a href="#"><u>RecChanged</u></a>	<a href="#"><u>RecLink</u></a>	<a href="#"><u>RecSize</u></a>

### Cell Attributes

<a href="#"><u>CellData</u></a>	<a href="#"><u>CellText</u></a>
---------------------------------	---------------------------------

## AutoExtend Attribute

When set, the Grid functions in a data-entry mode, with a blank record at the bottom.

### Usage

#### C/C++

Window Style: [HGS\\_AUTOEXTEND](#)

#### VBX

```
[form.][control.]AutoExtend[= True/False]
```

### Remarks

AutoExtend can only be set at design time.

The [RecNew](#) attribute, for the blank record will remain set until its contents have been edited. In [buffered grids](#), the blank record at the bottom will only be updated if its contents have been edited.

### See Also

[Bottom](#) Event

## Background Attribute

The color or pattern used to paint the background of the grid control

### Usage

#### C

```
hbrBkgnd = (HBRUSH)SendMessage(hWnd, HGM_GETBKGNDBRUSH, 0, 0L);  
hbrOldBkgnd = (HBRUSH)SendMessage(hWnd, HGM_SETBKGNDBRUSH,  
    (WPARAM)hbrNewBrush, 0L);
```

#### C++

##### OWL

```
hbrBkgnd = [THGridObj.]GetBkgndBrush(void);  
hbrOldBkgnd = [THGridObj.]SetBkgndBrush(hbrNewBrush);
```

##### MFC

```
pBkgnd = [CHGridObj.]GetBkgndBrush(void);  
pOldBkgnd = [CHGridObj.]SetBkgndBrush(pNewBrush);
```

#### VBX

```
[form.][control.]BackColor[ = color ]
```

See *Visual Basic Language Reference*, "BackColor, ForeColor Properties"

### Arguments/Parameters

HBRUSH hbrNewBrush	Handle of a new brush to be set as the background brush
CBrush pNewBrush	Pointer to a CBrush object containing the new background brush handle

### Return values

HBRUSH hbrBkgnd	Handle to the current background brush
HBRUSH hbrOldBkgnd	Handle to the previous background brush
CBrush *pBkgnd	Pointer to a CBrush object containing the current background brush handle
CBrush *pOldBkgnd	Pointer to a CBrush object containing the previous background brush handle

### Remarks

C and C++ applications are responsible for destroying any brushes they create.

## BaseRec Attribute

A long value that marks the starting position of the record buffer in the table. It is the first record of the table that is currently in memory.

### Usage

#### C

```
lBaseRec = SendMessage(hWnd, HGM_GETBASEREC, 0, 0L);
```

#### C++

```
lBaseRec = [CHGridObj.]GetBaseRec();
```

#### VBX

```
[form.][control.]BaseRec
```

### Return values

LONG lBaseRec

The current BaseRec value

### Remarks

For Grids with no record buffer, BaseRec is always 0 since all records are in memory. The BaseRec value can be added to a row index to obtain a record index. In fact, the RowToRec method does exactly this.

BaseRec should not be adjusted by an application; use the FirstRec instead.

## Browse Attribute

When Browse is set, the Grid displays data, but will not allow editing.

### Usage

#### C/C++

Window Style: HGS\_BROWSE

#### VBX

[*form.*][*control.*]**Browse**[= *True/False*]

### Remarks

Browse can only be set at design time.

## BtnHeight Attribute

The height, in pixels, of the ColButtons displayed along the top of the Grid

### Usage

#### C

```
wBtnHeight = (WORD)SendMessage(hWnd, HGM_GETBTNHEIGHT, 0, 0L);  
SendMessage(hWnd, HGM_SETBTNHEIGHT, wNewHeight, (LPARAM)bRedraw);
```

#### C++

```
wBtnHeight = [CHGridObj.]GetBtnHeight();  
[CHGridObj.]SetBtnHeight(wNewHeight[ , bRedraw = TRUE]);
```

#### VBX

```
[form.][control.]ButtonHeight[= integer]
```

### Arguments/Parameters

WORD wNewHeight	The new ButtonHeight value
BOOL bRedraw	TRUE forces the control to redraw immediately

### Return values

WORD wBtnHeight	The current ButtonHeight value
-----------------	--------------------------------

## BtnWidth Attribute

The width of the grids [RowBtns](#)

### Usage

#### C

```
iBtnWidth = (int)SendMessage(hWnd, HGM_GETBTNWIDTH, 0, 0L);  
SendMessage(hWnd, HGM_SETBTNWIDTH, wNewWidth, (LPARAM)bRedraw);
```

#### C++

```
iBtnWidth = [CHGridObj.]GetBtnWidth();  
[CHGridObj.]SetBtnWidth(wNewWidth [, bRedraw = TRUE]);
```

#### VBX

```
[form.][control.]BtnWidth[= integer]
```

### Arguments/Parameters

WORD wNewWidth

The new ButtonWidth value

BOOL bRedraw

TRUE forces the control to redraw immediately

### Return values

int iBtnWidth

The current ButtonWidth value

### See Also

[BtnHeight](#)

## BufferProc Attribute

A callback procedure(s) that is (are) responsible for retrieving records for a record buffer from a data source and maintaining data integrity at the data source.

### Usage

#### C

```
lpfnBufferProc = SendMessage(hWnd, HGM_GETBUFFERPROC, 0, 0L);  
lpfnOldBufferProc = SendMessage(hWnd, HGM_SETBUFFERPROC, 0,  
lpfnNewBufferProc);
```

#### C++

```
[CHGridBufferObj].AttachBuffer(lpGridObj);
```

#### VBX

Not Used

*Users of Visual Basic 3.0 may take advantage of the [Data Awareness](#) of the VBX Grid, which includes automatic record buffering.*

### Arguments/Parameters

<code>BUFFERPROC lpfnNewBufferProc</code>	A pointer to an application-defined record buffering callback procedure
<code>[CHGridObj] FAR *lpGridObj</code>	A pointer to a C++ Grid or Grid-derived object

### Return values

<code>BUFFERPROC lpfnBufferProc</code>	A pointer to the current record buffering callback procedure or NULL if no procedure is installed
<code>BUFFERPROC lpfnOldBufferProc</code>	A pointer to a previously installed record buffering callback procedure or NULL

### Remarks

For a complete discussion of record buffering in the Grid, see the section titled [Using a Record Buffer](#).

## BufferSize Attribute

An integer value representing the total number of records that the HGrid keeps in memory

### Usage

#### C

```
iBufSize = (int)SendMessage(hWnd, HGM_GETBUFFERSIZE, 0, 0L);  
iBufSize = (int)SendMessage(hWnd, HGM_SETBUFFERSIZE, 0,  
(LPARAM) iNewBufSize);
```

#### C++

```
iBufSize =[CHGridObj.]GetBufferSize();  
iBufSize =[CHGridObj.]SetBufferSize(iNewBufSize);
```

#### VBX

```
[form.][control.]BufferSize[= integer]
```

### Arguments/Parameters

<code>int iNewBufSize</code>	A new value for BufferSize
------------------------------	----------------------------

### Return values

<code>int iBufSize</code>	The current value for BufferSize or zero if the Grid is non-buffering
---------------------------	---

### Remarks

The BufferSize attribute can be set by an application in order to optimize performance. However, a minimum BufferSize of twice the number of visible records will always be maintained.

### See Also

[Using a Grid Buffer](#)



## ColButtons Attribute

When set, the Grid will display buttons containing field names at the top of each column.

### Usage

#### C/C++

Window Style: HGS\_COLBUTTONS

#### VBX

```
[form.][control.]ColButtons[= True/False]
```

### Arguments/Parameters

### Return values

### Remarks

The ColButtons attribute can only be set at design time.

### See Also

RowBtns

## ColCount Attribute

The number of columns in the Grid

### Usage

#### C

```
wColCount = SendMessage(hWnd, HGM_GETCOLCOUNT, 0, 0L);
```

#### C++

```
wColCount = [CHGridObj.]GetColCount();
```

#### VBX

```
[form.][control.]ColCount
```

### Return values

WORD wColCount

The current number of columns in the Grid

### Remarks

The value of the ColCount attribute includes hidden columns.

## ColMap Attribute

An array of integers showing the relationship between Field indices and Column indices, which may differ if [DragCols](#) is enabled

### Usage

#### C

```
SendMessage(hWnd, HGM_GETCOLMAP, 0, (LPARAM) lpMap);  
SendMessage(hWnd, HGM_SETCOLMAP, 0, (LPARAM) lpMap);
```

#### C++

```
[CHGridObj.]GetColMap(lpMap);  
[CHGridObj.]SetColMap(lpMap);
```

#### VBX

```
ReDim iaMap(Grid1.ColCount - 1) As Integer  
VGGetColMap([control.]hWnd, iaMap)  
VGSetColMap([control.]hWnd, iaMap)  
iaMap MUST contain exactly one value for every field
```

### Arguments/Parameters

int FAR \*lpMap

A pointer to an array of integers to be copied or filled by the control. The array **MUST** have space for one integer per field. Storing the MAP allows you to create custom configurations.

### Remarks

A sample ColMap is shown below:

**Your Record:** Field0 Field1 Field2 Field3 Field4 Field5 Field6 Field7 Field8

**ColMap:** 0 3 4 2 7 6 8 1 5

**Grid Display:** Field0 Field7 Field3 Field1 Field2 Field8 Field5 Field4 Field6

The ColMap contains an integer *column index* for each field in the record. The column index is the position of the field as displayed in the Grid, the field index is the position of the field in the data source for the control.

The user can select a column(s) then with the Shift key pressed, drag the column(s) to a new position. The current drop position is displayed as a thick column line. This does not affect the order of the data in the records. However, you will have to be aware that the columns may have moved when you go to select a cell or range of cells. Positions for selection are given by column index, not field index. Positions for setting Field attributes are still specified by field index.

If you insert a field using the InsertFld method, insert it at the column index equal to its field index then use the MoveFld method to reposition it within the Grid. When deleting a field, use the column index.

### See Also

[ColToFld](#), [FldToCol](#)

## DisableNoScroll Attribute

When DisableNoScroll is set, which it is by default, the vertical scroll bar is disabled but not hidden when its scroll range is empty.

### Usage

#### C/C++

Window Style: HGS\_DISABLENOSCROLL

#### VBX

[*form.*][*control.*]**DisableNoScroll**[= *TRUE/FALSE*]

### Remarks

The DisableNoScroll attribute can only be set at design time.

### See Also

VScroll, VScrollPos

## DragCols Attribute

When this attribute is set, the user can drag non-frozen columns to other locations in the Grid

### Usage

#### C/C++

Window Style: HGS\_DRAGCOLS

#### VBX

`[form.][control.]DragCols [= TRUE/FALSE]`

### Remarks

A user can move a column by clicking on its ColButton with the Shift key depressed and holding the mouse button down while dragging to the new location.

### See Also

ColMap

### Remarks

The DragCols attribute can only be set at design time.

## EditInPlace Attribute

When the EditInPlace attribute is set, user editing of Grid data occurs at each cell location rather than in a toolbar at the top of the grid.

### Usage

#### C/C++

Window Style: HGS\_INPLACE

#### VBX

[*form.*] [*control.*] **EditInPlace** [= *TRUE/FALSE*]

### Remarks

The EditInPlace attribute can only be set at design time.

## FirstCol Attribute

The index of the first visible, non-frozen Column

### Usage

#### C

```
wFirstCol = (WORD)SendMessage(hWnd, HGM_GETFIRSTCOL, 0, 0L);  
SendMessage(hWnd, HGM_SETFIRSTCOL, iCol, 0L);
```

#### C++

```
wFirstCol = [CHGridObj.]GetFirstCol();  
[CHGridObj.]SetFirstCol(iCol)
```

#### VBX

```
[form.][control.]FirstCol[= integer]
```

### Arguments/Parameters

int iCol	The index of the new FirstCol
----------	-------------------------------

### Return values

WORD wFirstCol	The index of the current FirstCol
----------------	-----------------------------------

### Remarks

Setting the value of the FirstCol attribute scrolls the table horizontally to bring the new column into the leftmost position in the Grid but to the right of any FrozenCols.

### See Also

FirstRec, HScrollPos, FrozenCols

## FirstRec Attribute

A long value that marks the first record of the table to be displayed in the HGrid

### Usage

#### C

```
lFirstRec = SendMessage(hWnd, HGM_GETFIRSTREC, 0, 0L);  
lFirstRec = SendMessage(hWnd, HGM_SETFIRSTREC, 0, lNewFirstRec);
```

#### C++

```
lFirstRec = [CHGridObj.]GetFirstRec();  
lFirstRec = [CHGridObj.]SetFirstRec(lNewFirstRec);
```

#### VBX

```
[form.][control.]FirstRec[= long]
```

### Arguments/Parameters

LONG lNewFirstRec	The new FirstRec value.
-------------------	-------------------------

### Return values

LONG lFirstRec	The sum of <u>BaseRec</u> and the row index of the first visible record.
----------------	--

### Remarks

An application can manipulate the viewport by setting the FirstRec attribute. If the FirstRec attribute is set to a position outside the current buffer, the buffer moves automatically by adjusting the BaseRec.

### See Also

[BaseRec](#)



## Font Attribute

The font used by the control

### Usage

#### C

```
hfFont = (HFONT)SendMessage(hWnd, HGM_GETFONT, 0, 0L);  
hfOldFont = SendMessage(hWnd, HGM_SETFONT, (WPARAM)hfNewFont,  
(LPARAM)bRedraw);
```

#### C++

##### OWL

```
hfFont = [THGridObj.]GetFont();  
hfOldFont[THGridObj.]SetFont(hfNewFont [, bRedraw = TRUE] );
```

##### MFC

```
pFont = [CHGridObj.]GetFont();  
pOldFont[CHGridObj.]SetFont(pNewFont [, bRedraw = TRUE] );
```

#### VBX

```
[form.][control.]FontBold[= boolean]  
[form.][control.]FontItalic[= boolean]  
[form.][control.]FontName[= font]  
[form.][control.]FontSize[= points]  
[form.][control.]FontStrikethru[= boolean]  
[form.][control.]FontUnderline[= boolean]
```

*See Visual Basic Language Reference, "FontName Property"*

### Arguments/Parameters

HFONT hfNewFont	Handle of the font to be set
BOOL bRedraw	A value of TRUE causes the control to repaint immediately
CFont *pFont	Pointer to a CFont object containing the handle to the font to be set

### Return values

HFONT hfFont	Handle to the control's current font
HFONT hfOldFont	Handle to the control's previous font
CFont *hfFont	Pointer to a CFont object containing the handle to the control's current font
CFont *fOldFont	Pointer to a CFont object containing the handle to the control's previous font

### Remarks

C and C++ applications are responsible for destroying any fonts they create.

## FrozenCols Attribute

The number of columns on the left side of the Grid that do not scroll horizontally

### Usage

#### C

```
wFrozenFlds = (WORD)SendMessage(hWnd, HGM_GETFROZENCOLS, 0, 0L);  
SendMessage(hWnd, HGM_SETFROZENCOLS, wNumFrozen, 0L);
```

#### C++

```
wFrozenFlds = [CHGridObj.]GetFrozenCols();  
[CHGridObj.]SetFrozenCols(wNumFrozen);
```

#### VBX

```
[form.][control.]FrozenCols [= integer]
```

### Arguments/Parameters

WORD wNumFrozen	The new number of frozen fields
-----------------	---------------------------------

### Return values

WORD wFrozenFlds	The current number of frozen fields
------------------	-------------------------------------

## HScroll Attribute

When HScroll is set, which it is by default, a horizontal scrollbar will appear whenever the total width of the columns exceeds that of the Grid's display area.

### Usage

#### C/C++

Window Style: WS\_HSCROLL

#### VBX

[*form.*][*control.*]**HScroll**[= *TRUE/FALSE*]

### Remarks

The HScroll attribute can only be set at design time.

## HScrollPos Attribute

The horizontal scroll position of the Grid within its scroll range. The range is equal to the number of non-frozen columns that must be scrolled off the display area in order to make the rightmost column completely visible.

### Usage

#### C

```
iHScroll = (int)SendMessage(hWnd, HGM_GETHSCROLLPOS, 0, 0L);  
SendMessage(hWnd, HGM_SETHSCROLLPOS, iNewHScroll, 0L);
```

#### C++

```
iHScroll = [CHGridObj.]GetHScrollPos();  
[CHGridObj.]SetHScrollPos(iNewHScroll);
```

#### VBX

```
[form.][control.]HScrollPos[ = iNewHScroll]
```

### Arguments/Parameters

<code>int iNewHScroll</code>	The new horizontal scroll position
------------------------------	------------------------------------

### Return values

<code>int iHScroll</code>	The current horizontal scroll position
---------------------------	--

## KbdDelIns Attribute

When KbdDelIns is set, users can delete and insert records using the Delete and Insert keys at run time.

### Usage

#### C/C++

Window Style: HGS\_KBDELINS

#### VBX

[*form.*][*control.*]KbdDelIns [= TRUE/FALSE]

### Remarks

The KbdDelIns attribute can only be set at design time.

## LeaveOnTab Attribute

When LeaveOnTab is set, the Grid loses focus when the Tab key is pressed

### Usage

#### C/C++

Window Style: HGS\_LEAVEONTAB

#### VBX

[*form.*] [*control.*] **LeaveOnTab**

### Remarks

The LeaveOnTab attribute can only be set at design time.

## Marker Attribute

Temporary Row and Column coordinates maintained by the Grid for the purpose of communicating cell locations between the application and the control.

### Usage

#### C

```
lMarker = SendMessage(hWnd, HGM_GETMARKER, 0, 0L);  
SendMessage(hWnd, HGM_SETMARKER, 0, MAKELONG(iCol, iRow));
```

#### C++

```
lMarker = [CHGridObj.]GetMarker();  
[CHGridObj.]SetMarker(iCol, iRow);
```

#### VBX

```
[form.][control.].Col[= iCol]  
[form.][control.].Row[= iRow]
```

### Arguments/Parameters

int iCol  
int iRow

The column index of the cell  
The row index of the cell

### Return Value

LONG lMarker

Contains an integer column index in the LOWORD  
and an integer row index in the HIWORD

### Remarks

The Marker is set by the Grid to the new coordinates prior to a [SelChanging](#) or [SelExtending](#) event. Changing the Marker in response to these notifications changes the resulting selection. In the VBX Grid, the Marker must be set by the application prior to setting the [CellData](#) property in order to specify which cell's data to set.

## MaxRec Attribute

Long value representing the total number of records in the table

### Usage

#### C

```
lMaxRec = SendMessage(hWnd, HGM_GETMAXREC, 0, 0L);  
bSuccess = (BOOL)SendMessage(hWnd, HGM_SETMAXREC, 0, lNewMaxRec);
```

#### C++

```
lMaxRec = [CHGridObj.]GetMaxRec();  
bSuccess = [CHGridObj.]SetMaxRec(lNewMaxRec);
```

#### VBX

```
[form.][control.]MaxRec[= long]
```

### Arguments/Parameters

LONG lNewMaxRec	The new value for MaxRec
-----------------	--------------------------

### Return values

LONG lMaxRec	The current value for MaxRec
BOOL bSuccess	TRUE if a new MaxRec value was set successfully

### Remarks

The MaxRec attribute is typically used for Grids with record buffers, where the entire record space is not present in the control's memory at any given time. If MaxRec is set by an application, the HGrid will provide a virtual scroll range from zero to MaxRec. If MaxRec is not set, it will grow automatically when records are added and/or the BaseRec changes. If the Grid has the AutoExtend attribute set, no new records will be added beyond MaxRec

### See Also

[Using a Grid Buffer](#)



## MDIChild Attribute

When set, the control will be created as an MDI child window

### Usage

#### C/C++

Window Style: HGS\_MDICHILD

#### VBX

[*form.*][*control.*]**MDIChild**[= *True/False*]

### Remarks

C++ developers using the MFC wrapper classes should use the CHGridView class to create MDI applications with Grid windows.

The MDIChild attribute can only be set at design time.

## NoHideSel Attribute

When NoHideSel is set, the Grid continues to display the current Selection after losing focus

### Usage

#### C/C++

Window Style: HGS\_NOHIDSEL

#### VBX

[*form.*][*control.*]**NoHideSel** [= TRUE/FALSE]

### Remarks

The NoHideSel attribute can only be set at design time.

## NoLines Attribute

When NoLines is set, the Grid is displayed without lines separating cells

### Usage

#### C/C++

Window Style: HGS\_NOLINES

#### VBX

[*form.*] [*control.*] **NoLines**

### Remarks

The NoLines attribute can only be set at design time.

## Quiet Attribute

When the control is in Quiet mode, it does not send notification messages to its parent. VBX controls will not fire events in Quiet mode.

### Usage

#### C

```
bIsQuiet = SendMessage(hWnd, HGM_ISQUIET, 0, 0L);  
SendMessage(hWnd, HGM_BEQUIET, bValue, 0L);
```

#### C++

```
bIsQuiet = [CHGridObj.]IsQuiet  
[CHGridObj.]BeQuiet(bValue);
```

#### VBX

```
bIsQuiet = SendMessage(control.hWnd, HGM_ISQUIET, 0, 0L);  
SendMessage(control.hWnd, HGM_BEQUIET, bValue, 0L)
```

See [VBX Advanced Topics](#)

### Arguments/Parameters

BOOL bValue

TRUE turns on Quiet mode, FALSE turns it off

### Return values

BOOL bIsQuiet

TRUE if the control is in Quiet mode

## ResizeCols Attribute

When ResizeCols is set, columns can be resized horizontally by the user at runtime

### Usage

#### C/C++

Window Style: HGS\_RESIZECOLS

#### VBX

[*form.*][*control.*]**ResizeCols**[= *TRUE/FALSE*]

### Remarks

The ResizeCols attribute can only be set at design time.

## ResizeRows Attribute

When set, the rows of the grid can be resized by the user at run time.

### Usage

#### C/C++

Window Style: HGS\_RESIZEROWS

#### VBX

[*form.*][*control.*]**ResizeRows**[= *True/False*]

### Remarks

The ResizeRows attribute can only be set at design time.

## RowBtns Attribute

When the RowBtns attribute is set, the control displays buttons to the left of each row.

### Usage

#### C/C++

Window Style: HGS\_ROWBTNS

#### VBX

[*form.*] [*control.*] **RowBtns**

### Remarks

The number displayed in a row button is the *record* index plus one. If a record buffer is used, record indices should not be confused with row indices. Record indices range from zero to the value of MaxRec minus one, whereas row indices range from zero to the value of BufferSize minus one. If no buffer is used, row index and record index are the same.

### See Also

ColButtons

## RowCount Attribute

The number of rows in the Grid.

### Usage

#### C

```
iRowCount = SendMessage(hWnd, HGM_GETROWCOUNT, 0, 0L);
```

#### C++

```
iRowCount = [CHGridObj.]GetRowCount();
```

#### VBX

```
[form.][control.]RowCount[= iRowCount]
```

### Return values

int iRowCount

The current number of rows in the Grid

### Remarks

If a record buffer is used, RowCount refers to the number of records in the buffer. RowCount is equal to BufferSize if the buffer is full.



## RowHeight Attribute

The height, in pixels, of the rows in the Grid

### Usage

#### C

```
wRowHeight = SendMessage(hWnd, HGM_GETROWHEIGHT, 0, 0L);  
SendMessage(hWnd, HGM_SETROWHEIGHT, wNewHeight, (LPARAM)bRedraw);
```

#### C++

```
wRowHeight = [CHGridObj.]GetRowHeight();  
[CHGridObj.]SetRowHeight(wNewHeight[ , bRedraw = TRUE]);
```

#### VBX

```
[form.][control.]RowHeight[= integer]
```

### Arguments/Parameters

WORD wNewHeight	The new RowHeight value
BOOL bRedraw	TRUE forces the control to redraw immediately

### Return values

WORD wRowHeight	The current RowHeight value
-----------------	-----------------------------

### Remarks

All rows in the Grid have the same height.

## Selection Attribute

Row and column indices that define the range of selected cells in the Grid

### Usage

#### C

```
lCell = SendMessage(hWnd, HGM_GETSELANCHOR, 0, lpCell);  
lCell = SendMessage(hWnd, HGM_GETSELEXTENT, 0, lpCell);  
SendMessage(hWnd, HGM_SETSELEXTENT, (WPARAM)bExtend, MAKELONG(iCol,  
iRow));
```

#### C++

```
lCell = [CHGridObj.]GetSelAnchor( [lpCell = NULL] );  
[CHGridObj.]SetSelAnchor(iCol, iRow);  
lCell = [CHGridObj.]GetSelExtent( [lpCell = NULL] );  
[CHGridObj.]SetSelExtent(bExtend, iCol, iRow);
```

#### VBX

The following properties are read-only:

```
[form.][control.]SelStartCol  
[form.][control.]SelStartRow  
[form.][control.]SelEndCol  
[form.][control.]SelEndRow
```

Use these methods to set the selection:

```
VGSelectCell([control.]hWnd, iCol, iRow)  
VGSelectRange([control.]hWnd, iCol1, iRow1, iCol2, iRow2)
```

### Arguments/Parameters

LONG FAR *lpCell	A pointer to a long integer that will receive a column index in the LOWORD and a row index in the HIWORD. lpCell can be NULL.
int iCol	A column index
int iRow	A row index
BOOL bExtend	TRUE extends the selection from the Anchor; FALSE sets <i>both</i> Anchor and Extent to (iCol, iRow).

### Return values

LONG lCell	Used as an alternative to lpCell. A long value having a column index in the LOWORD and a row index in the HIWORD.
------------	---

### Remarks

The *Selection Anchor* is the cell at which a selection was begun; the *Selection Extent* is the cell at which a selection ends. The cell indicated by the Selection Extent is also the location of the cursor. If a single cell is selected by setting the Extent with bExtend = FALSE.

RowButtons on the left of the Grid have a *column* index of -1. If the column index in the LOWORD of Selection Extent is -1, then the entire row given by the row index is selected. Column Buttons along the top of the Grid have a *row* index of -1. If the row index in the HIWORD of Selection Extent is -1, then the entire row given by the row index is selected.

A Selection Extent of (-1, -1) is the button in the top left-hand corner of the Grid. Setting Selection Extent to (-1, -1) selects the entire grid.

**See Also**

[ColToFld](#), [RowToRec](#)

## SingleSelect Attribute

When SingleSelect is set, the user may select only individual cells, not ranges.

### Usage

#### C/C++

Window Style: HGS\_SINGLESELECT

#### VBX

[*form.*] [*control.*] **SingleSelect**

### See Also

WholeRows

## State Attribute

A collection of flags describing the state of the control

### Usage

#### C

```
lState = SendMessage(hWnd, HGM_GETSTATE, 0, 0L);  
SendMessage(hWnd, HGM_SETSTATE, (WPARAM)bValue, (LPARAM)wStateFlag);
```

#### C++

```
lState = [CHGridObj.]GetState();  
[CHGridObj.]SetState(bValue, wStateFlag);
```

#### VBX

```
[form.][control.]Changed[= boolean]
```

### Arguments/Parameters

BOOL bValue

The value to set for the flag indicated by wStateFlag

WORD wStateFlag

One of the [Grid state flag constants](#)

### Return values

LONG lState

The current State flags. Individual flag settings can be obtained by AND'ing lState with a state flag constant

## Style Attribute

A collection of bits used to determine various Grid attributes.

### Usage

#### C

```
lStyle = SendMessage(hWnd, HGM_GETSTYLE, 0, 0L);  
SendMessage(hWnd, HGM_SETSTYLE, bValue, (LPARAM)lFlag);
```

#### C++

```
lStyle = [CHGridObj.]GetStyle();  
[CHGridObj.]SetStyle(bValue, lFlag);
```

#### VBX

Not Used

### Arguments/Parameters

LONG lStyle

The current style bits for the Grid. This can be bitwise-and'ed with a particular Grid Style Flag to determine that flags setting.

BOOL bValue

The new value for a particular style bit

LONG lFlag

One of the Grid Style Flags.

## Title Attribute

A character string to be used in the Grid window caption if it is displayed

### Usage

#### C

```
SendMessage(hWnd, HGM_GETTITLE, wCount, (LPARAM)lpBuf);  
SendMessage(hWnd, HGM_SETTITLE, 0, (LPARAM)lpTitle);
```

#### C++

```
[CHGridObj.]GetTitle(wCount, lpBuf);  
[CHGridObj.]SetTitle(lpTitle);
```

#### VBX

```
[form.][control.]Caption[ = stringexpression]  
See Visual Basic Language Reference, "Caption Property"
```

### Arguments/Parameters

WORD wCount	The number of characters to copy to or from lpBuf
LPSTR lpBuf	A buffer to hold the Title
LPSTR lpTitle	The new Title string

## VScroll Attribute

When VScroll is set, which it is by default, a vertical scrollbar will appear whenever the total height of the records exceeds that of the Grid's display area.

### Usage

#### C/C++

Window Style: WS\_VSCROLL

#### VBX

[*form.*][*control.*]**VScroll**[= *TRUE/FALSE*]

### Remarks

The VScroll attribute can only be set at design time.



## VScrollPos Attribute

The vertical scroll position of the Grid within its scroll range. The range is equal to the number of records that must be scrolled off the display area in order to make the bottom record completely visible.

### Usage

#### C

```
iVScroll = (int)SendMessage(hWnd, HGM_GETVSCROLLPOS, 0, 0L);  
SendMessage(hWnd, HGM_SETVSCROLLPOS, 0, (LPARAM)iNewVScroll);
```

#### C++

```
iVScroll = [CHGridObj.]GetVScrollPos();  
[CHGridObj.]SetVScrollPos(iNewVScroll);
```

#### VBX

```
[form.][control.]VScrollPos[ = iNewVScroll]
```

### Arguments/Parameters

<code>int iNewVScroll</code>	The new vertical scroll position
------------------------------	----------------------------------

### Return values

<code>int iVScroll</code>	The current vertical scroll position
---------------------------	--------------------------------------

## WholeRows Attribute

When WholeRows is set, selecting an individual cell sets the selection to the entire row.

### Usage

#### C

**Window Style:** `HGS_WHOLEROWS`

```
iSelected = SendMessage(hWnd, HGM_GETSELCOUNT, 0, 0L);
SendMessage(hWnd, HGM_GETSELROWS, iBufSize, lpBuffer);
SendMessage(hWnd, HGRM_ISSELECTED, iRow, 0L);
SendMessage(hWnd, HGM_SELECTROW, iRow, 0L);
```

#### C++

**Window Style:** `HGS_WHOLEROWS`

```
iSelected = [CHGridObj.]GetSelCount();
[CHGridObj.]GetSelRows(iBufSize, lpBuffer);
[CHGridObj.]IsSelected(iRow);
[CHGridObj.]SelectRow(iRow);
```

#### VBX

```
[form.][control.]WholeRows
[form.][control.]RowSelected(iRow) [ = boolean]
```

### Arguments/Parameters

int iBufSize	The size in bytes of lpBuffer
int far *lpBuffer	An array of integers to hold selected row information.
int iRow	The row index.

### Return values

int iSelected	The number of selected rows.
---------------	------------------------------

### Remarks

WholeRow selection mode also supports selection of discontinuous ranges of rows. Holding down the Ctrl key while clicking or dragging with the mouse selects additional rows or ranges of rows without deselecting previously selected ones. *Note: This feature is new as of version 2.01. Old applications using WholeRows selection can no longer use the [selection anchor](#) and [selection extent](#) to determine the currently selected rows. This will now only return the last contiguous range selected. The messages and methods listed in this topic will allow the selection to be checked and changed*

### See Also

[SingleSelect](#)

## FldCodeClass Attribute

One of the data classes defined in the [Data Engine](#) chapter

### Usage

#### C

```
cCodeClass = (char)SendMessage(hWnd, HGFM_GETCODECLASS, iFld, 0L);
```

#### C++

```
cCodeClass = [CHGridObj.]GetCodeClass(iFld);
```

#### VBX

```
[form.][control.]FldCodeClass(iFld)
```

### Arguments/Parameters

int iFld	The field index
----------	-----------------

### Return values

char cCodeClass	One of the <a href="#">data class character codes</a>
-----------------	---

### Remarks

The FldCodeClass attribute can only be set at design time. Control over this attribute is provided by the control design dialogs in Dialog Editor and Resource Workshop, HGEEdit, and the Properties dialog in Visual Basic. When creating fields dynamically in C and C++ using the CreateField method, the DataClass is included as an argument.

### See Also

[FldCodeType](#)

## FldCodeType Attribute

One of the data types defined in the [DataEngine](#) chapter.

### Usage

#### C

```
cCodeType = (char)SendMessage(hWnd, HGFM_GETCODETYPE, iFld, 0L);
```

#### C++

```
cCodeType = [CHGridObj.]GetCodeType(iFld);
```

#### VBX

```
[form.][control.]FldCodeType(iFld)
```

### Arguments/Parameters

<code>int iFld</code>	The field index
-----------------------	-----------------

### Return values

<code>char cCodeType</code>	One of the <a href="#">data type character codes</a>
-----------------------------	--

### See Also

[FldCodeClass](#), [FldDataSize](#)

## FldColWidth Attribute

The width of the field column in the Grid

### Usage

#### C

```
iWidth = (int)SendMessage(hWnd, HGFM_GETCOLWIDTH, (WPARAM)iFld, 0L);  
SendMessage(hWnd, HGFM_SETCOLWIDTH, (WPARAM)iFld, (LPARAM)iNewWidth);
```

#### C++

```
iWidth = [CHGridObj.]GetColWidth(iFld);  
[CHGridObj.]SetColWidth(iFld, iNewWidth);
```

#### VBX

```
[form.][control.]FldColWidth(iFld)[= iWidth]
```

### Arguments/Parameters

<code>int iFld</code>	The field index
<code>int iNewWidth</code>	The new FldColWidth value

### Return values

<code>int iWidth</code>	The current FldColWidth value
-------------------------	-------------------------------

### See Also

[FldDropHeight](#)

## FldCtlStyle Attribute

The style bits for a given field's control

### Usage

#### C

```
lCtlStyle = SendMessage(hWnd, HGFM_GETCTLSTYLE, (WPARAM)iFld, 0L);
```

#### C++

```
lCtlStyle = [CHGridObj.]GetCtlStyle(iFld)
```

#### VBX

Not Used

### Arguments/Parameters

<code>int iFld</code>	The field index of the field
-----------------------	------------------------------

### Return values

<code>LONG lCtlStyle</code>	The style bits set for
-----------------------------	------------------------

### See Also

[CreateField](#)

## FldCtlType Attribute

The type of control (Edit, List, etc.) responsible for handling interaction for a given field

### Usage

#### C

```
cCtlType = SendMessage(hWnd, HGFM_GETCTLTYPE, (WPARAM)iFld, 0L);
```

#### C++

```
cCtlType = [CHGridObj.]GetCtlType(iFld)
```

#### VBX

```
[CHGridObj.]FldCtlType
```

### Arguments/Parameters

int iFld

The field index of the field

### Return values

char cCtlType

One of the [control type constants](#)

### See Also

[CreateField](#)

## FldDataClass Attribute

One of the data classes defined in the [Data Engine](#) chapter.

### Usage

#### C

```
cFldDataClass = (char)SendMessage(hWnd, HGFM_GETDATACLASS, iFld, 0L);
```

#### C++

```
cFldDataClass = [CHGridObj.]GetDataClass(iFld);
```

#### VBX

```
[form.][control.]DataClass
```

### Arguments/Parameters

<code>int iFld</code>	The field index of the field
-----------------------	------------------------------

### Return values

<code>char cDataClass</code>	One of the <a href="#">data class character codes</a>
------------------------------	---

### Remarks

The FldDataClass attribute is read-only at run time. Control over this attribute is provided by the control design dialogs in Dialog Editor and Resource Workshop, HGEEdit, and by the Properties dialog in Visual Basic. When creating fields dynamically in C and C++ using the [CreateField](#) method, the DataClass is included as an argument.

### See Also

[FldDataType](#)



## FldDataSize Attribute

The size of the data for a field in bytes. This is may be the size of the Codes for a ListBox or DropList field that employs Codes.

### Usage

#### C

```
iSize = (int)SendMessage(hWnd, HGFM_GETDATASIZE, iFld, 0L);
```

#### C++

```
iSize = [CHGridObj.]GetDataSize(iFld);
```

#### VBX

```
[form.][control.]FldDataSize(iFld)
```

### Arguments/Parameters

<code>int iFld</code>	The field index of the field
-----------------------	------------------------------

### Return values

<code>int iSize</code>	The size of the data in bytes.
------------------------	--------------------------------

## FldDataType Attribute

One of the data types defined in the [DataEngine](#) chapter.

### Usage

#### C

```
cDataType = (char)SendMessage(hWnd, HGFM_GETDATATYPE, iFld, 0L);
```

#### C++

```
cDataType = [CHGridObj.]GetDataType(iFld);
```

#### VBX

```
[form.][control.]FldDataType(iFld)
```

### Arguments/Parameters

int iFld	The field index
----------	-----------------

### Return values

char cDataType	One of the <a href="#">data type character codes</a>
----------------	--

### See Also

[FldDataClass](#), [FldDataSize](#)

## FldDropHeight Attribute

The height of a ComboBox field's drop-down list box

### Usage

#### C

```
wHeight = (WORD)SendMessage(hWnd, HGFM_GETDROPHEIGHT, iFld, 0L);  
SendMessage(hWnd, HGFM_SETDROPHEIGHT, iFld, (LPARAM)wNewHeight);
```

#### C++

```
wHeight = [CHGridObj.]GetDropHeight(iFld);  
[CHGridObj.]SetDropHeight(iFld);
```

#### VBX

```
[form.][control.]FldDropHeight(iFld)
```

### Arguments/Parameters

int iFld	The field index of the field
WORD wNewHeight	The new value for FldDropHeight

### Return values

WORD wHeight	The current FldDropHeight value
--------------	---------------------------------

### See Also

[FldCtlType](#)

## FldFormat Attribute

A NULL-terminated character string that describes the way the Data is to be displayed

### Usage

#### C

```
IBytesCopied = SendMessage(hWnd, HGFM_GETFORMAT, (WPARAM)iFld,  
(LPARAM)lpszFormat);
```

#### C++

```
IBytesCopied = [CHGridObj.]GetFormat(iFld, lpszFormat);
```

#### VBX

```
[form.][control.]FldFormat[= iFld]
```

### Arguments/Parameters

<code>int iFld</code>	The field index
<code>LPSTR lpszFormat</code>	A pointer to a buffer for the FldFormat string

### Return values

<code>int iCopied</code>	The number of bytes actually copied to or from the buffer
--------------------------	---

### Remarks

An initial format string for fields created dynamically using [CreateField](#) is passed as an argument.

## FldFormatLen Attribute

The length in characters of a Field's format string

### Usage

#### C

```
wFormatLen = (WORD)SendMessage(hWnd, HGFM_GETFORMATLEN, iFld, 0L);
```

#### C++

```
wFormatLen = [CHGridObj.]GetFormatLen(iFld);
```

#### VBX

```
Len([form.][control.]FldFormat(iFld))
```

### Arguments/Parameters

<code>int iFld</code>	The field index of the field
-----------------------	------------------------------

### Return values

<code>WORD wFormatLen</code>	The value of FldFormatLen at iFld
------------------------------	-----------------------------------

### See Also

[FldFormat](#)

## FldName Attribute

A character string associated with a field that is displayed in a [ColButton](#) and can be used to locate a field in the Grid

### Usage

#### C

```
bSuccess = (int)SendMessage(hWnd, HGFM_GETNAME, iFld, lpszName);
```

```
bSuccess = (int)SendMessage(hWnd, HGFM_SETNAME, iFld, lpszName);
```

#### C++

```
bSuccess = [CHGridObj.]GetName(iFld, lpszName);
```

```
bSuccess = [CHGridObj.]SetName(iFld, lpszName);
```

#### VBX

```
[form.][control.]FldName(iFld) [= lpszName]
```

### Arguments/Parameters

int iFld

The field index of the field

LPSTR lpszName

A pointer to a buffer to hold the field name, or a pointer to the buffer that contains the new field name.

### Return values

BOOL bSuccess

TRUE if name of iFld was retrieved or set successfully.

### See Also

[FldNameLen](#), [FindFld](#)

## FldNameLen Attribute

The length, in characters, of a field's FldName attribute

### Usage

#### C

```
iNameLen = SendMessage(hWnd, HGFM_GETNAMELEN, (WPARAM)iFld, 0L);
```

#### C++

```
iNameLen = [CHGridObj.]GetNameLen(iFld)
```

#### VBX

```
Len ([form.][control.]FldName(iFld))
```

### Arguments/Parameters

<code>int iFld</code>	The field index
-----------------------	-----------------

### Return values

<code>int iNameLen</code>	The length of FldName for iFld
---------------------------	--------------------------------

### See Also

[FldName](#)

## FldOffset Attribute

The byte offset into the record data structure for a field's data

### Usage

#### C

```
iOffset = SendMessage(hWnd, HGFM_GETOFFSET, (WPARAM)iFld, 0L);
```

#### C++

```
iOffset = [CHGridObj.]GetOffset(iFld);
```

#### VBX

Not Used

### Arguments/Parameters

<code>int iFld</code>	The field index of the field
-----------------------	------------------------------

### Return values

<code>int iOffset</code>	The value of FldOffset for iFld
--------------------------	---------------------------------



## FldState Attribute

A collection of flags describing the state of each field

### Usage

#### C

```
wState = SendMessage(hWnd, HGFM_GETSTATE, iFld, 0L);  
SendMessage(hWnd, HGFM_SETSTATE, iFld, MAKELONG(bValue, wStateFlag));
```

#### C++

```
wState = [CHGridObj.]GetFldState(iFld);  
[CHGridObj.]SetFldState(iFld, wStateFlag, bValue);
```

#### VBX

```
[form.][control.]FldBrowse(iFld) [= boolean]  
[form.][control.]FldHidden(iFld) [= boolean]
```

### Arguments/Parameters

int iFld	The field index
BOOL bValue	The value to set for the flag indicated by wStateFlag
WORD wStateFlag	On of the <u>Field state flag constants</u>

### Return values

WORD wState	The current State flags. Individual flag settings can be obtained by AND'ing IState with a state flag constant
-------------	--

## FldWindow Attribute

The window handle of an Edit Control, List, Combo Box, or Check Box that is responsible for handling user interaction and displaying data within a given Grid field

### Usage

#### C

```
hwControl = (HWND)SendMessage(hWnd, HGFM_GETHCTL, iFld, 0L);
```

#### C++

##### OWL

```
hwControl = [CHGridObj.]GetHctl(iFld);
```

##### MFC

```
pControl = [CHGridObj.]GetHctl(iFld);
```

#### VBX

```
[form.][control.]FldHWND(iFld)
```

### Arguments/Parameters

int iFld	A field index
----------	---------------

### Return values

HWND hwControl	A window handle for the field's control
CWnd *pControl	Pointer to a temporary CWnd-derived object corresponding to the <u>field control type</u> . Must be cast to the <u>correct object class</u>

### Remarks

Applications can often implement customized Grid behavior by sending messages directly to field controls.

## RecData Attribute

A Record data structure with an element for each field

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HGRM_GETDATA, wRowIndex,  
    (LPARAM) lpRecStruct);  
  
bSuccess = SendMessage(hWnd, HGRM_SETDATA, wRowIndex,  
    (LPARAM) lpRecStruct);
```

#### C++

```
bSuccess = [CHGridObj.]GetData(wRowIndex, lpRecStruct);  
bSuccess = [CHGridObj.]SetData(wRowIndex, lpRecStruct);
```

#### VBX

Not Used - see CellData

### Arguments/Parameters

WORD wRowIndex	The row at which to set the RecData
void FAR *lpRecStruct	A pointer to a record data structure

### Return values

BOOL bSuccess	TRUE if the RecodData was retrieved or set successfully.
---------------	--

### Remarks

The Visual Basic interface does not allow access to the RecData attribute directly. Users of the VBX WinWidgets can get and set data at individual cells by setting the CurrentRow and CurrentColumn and getting or setting the Text property. Users of Visual Basic 3.0 can alternately make use of the Grid's data awareness features to get and set data.

### See Also

RecLink

## RecLink Attribute

A pointer to a buffer which will be updated with the current contents of the currently selected record

### Usage

#### C

```
lpLink = (void FAR *)SendMessage(hWnd, HGRM_GETLINK, 0, 0L);  
bSuccess = (BOOL)SendMessage(hWnd, HGRM_SETLINK, 0, (LPARAM)lpBuf);
```

#### C++

```
lpLink = [CHGridObj.]GetRecLink();  
bSuccess = [CHGridObj.]SetRecLink(lpBuf);
```

#### VBX

Not Used

### Arguments/Parameters

void FAR \*lpBuf

A pointer to a buffer to which the control will copy the contents of the currently selected Record, or NULL to break an existing RecordLink.

### Return values

void FAR \*lpLink

BOOL bSuccess

The current RecordLink or NULL if no link exists  
TRUE if the RecordLink was installed successfully

### See Also

[RecData](#), [RecordStructures](#)

## RecSize Attribute

The size in bytes of a record data structure.

### Usage

#### C

```
wRecSize = (WORD)SendMessage(hWnd, HGRM_GETSIZE, 0, 0L);
```

#### C++

```
wRecSize = [CHGridObj.]GetRecSize();
```

#### VBX

Not Used

### Return values

WORD wRecSize

The current RecSize value

## RecState Attribute

A collection of flags describing the state of each record

### Usage

#### C

```
wState = SendMessage(hWnd, HGRM_GETSTATE, wRowIndex, 0L);  
SendMessage(hWnd, HGRM_SETSTATE, wRowIndex, MAKELONG(bValue,  
wStateFlag));
```

#### C++

```
wState = [CHGridObj.]GetRecState(wRowIndex);  
[CHGridObj.]SetRecState(wRowIndex, wStateFlag, bValue);
```

#### VBX

[*form.*][*control.*]**RecChanged**(*iRow*) [ = boolean]

indicates that some portion of the record's data has been altered

[*form.*][*control.*]**RecBrowse**(*iRow*) [ = boolean]

indicates that the record may not be edited by the user

[*form.*][*control.*]**RecNew**(*iRow*) [ = boolean]

indicates that the record was added via the keyboard insertion or via auto-extension

### Arguments/Parameters

WORD wIndex

The row index of the record

BOOL bValue

The value to set for the flag indicated by wStateFlag

WORD wStateFlag

One of the Record state flag constants

### Return values

WORD wState

The current State flags. Individual flag settings can be obtained by AND'ing IState with a state flag constant

## CellData Attribute

Data associated with a given cell of the Grid of a type defined in the [DataEngine](#) chapter

### Usage

#### C

```
bSuccess = HGGetCellData(hWnd, iCol, iRow, lpData);  
bSuccess = HGSetCellData(hWnd, iCol, iRow, lpData);  
bSuccess = HGSetCellString(hWnd, iCol, iRow,  
lpzStr);
```

#### C++

```
bSuccess = GetCellData(iCol, iRow, lpData);  
bSuccess = SetCellData(iCol, iRow, lpData);  
bSuccess = SetCellString(iCol, iRow, lpzStr);
```

#### VBX

```
[form.][control.]Data[= lpzStr]
```

### Arguments/Parameters

int iCol	The column index of the cell
int iRow	The row index of the cell
void FAR *lpData	A pointer to the data
LPSTR lpzStr	A pointer to a string used to be copied to a string field cell.

### Return values

BOOL bSuccess	TRUE if the CellData was retrieved or set correctly
---------------	---

### Remarks

The SetCellString methods set the data for a cell in a string field, using a null-terminated *strcpy()* rather than the fixed-length *memcpy()* of HGSetCellData.

The [Col](#) and [Row](#) properties in the VBX grid are used to select Grid coordinates at which the CellText attribute is to be set.

### See Also

[RecData](#)

## CellText Attribute

The formatted text for a given cell of the Grid

### Usage

#### C

```
bSuccess = HGGetCellText(hWnd, iCol, iRow, lpText, iMax);
```

#### C++

```
bSuccess = GetCellText(iCol, iRow, lpText, iMax);
```

#### VBX

Not Used

### Arguments/Parameters

<code>int iCol</code>	The Col index of the cell
<code>int iRow</code>	The row index of the cell
<code>LPSTR lpText</code>	A pointer to a buffer for the CellText
<code>int iMax</code>	The maximum number of characters to copy to lpText

### Return values

<code>BOOL bSuccess</code>	TRUE if the CellData was retrieved correctly
----------------------------	--

### See Also

[CellData](#)



## HGrid Methods

<u>AddFld</u>	<u>FindFld</u>	<u>MoveRec</u>
<u>AddRec</u>	<u>FldToCol</u>	<u>OffsetPtr</u>
<u>ColToFld</u>	<u>GetCurRec</u>	<u>RecToRow</u>
<u>CreateField</u>	<u>InsertFld</u>	<u>Reset</u>
<u>DeleteFld</u>	<u>InsertRec</u>	<u>RowToRec</u>
<u>DeleteRec</u>	<u>Invalidate</u>	<u>Update</u>
<u>DestroyField</u>	<u>MoveCol</u>	

## HGrid Record Management

<u>AddRec</u>	<u>MoveRec</u>
<u>DeleteRec</u>	<u>RecToRow</u>
<u>GetCurRec</u>	<u>Reset</u>
<u>InsertRec</u>	<u>RowToRec</u>
<u>Invalidate</u>	<u>Update</u>

## HGrid Field Management

<u>AddFld</u>	<u>FindFld</u>
<u>ColToFld</u>	<u>FldToCol</u>
<u>CreateField</u>	<u>InsertFld</u>
<u>DeleteFld</u>	<u>MoveCol</u>
<u>DestroyField</u>	

## AddFld Method

Appends a new Field to the Grid

### Usage

#### C

```
iFld = (int)SendMessage(hWnd, HGM_ADDFLD, 0, hFld);
```

#### C++

```
iFld = [CHGridObj.]AddFld(hFld);
```

#### VBX

```
iFld = VGAddFld([control.]hWnd, hFld)
```

### Arguments/Parameters

HFLD hFld

The 32-bit handle of a field structure returned by the CreateField method

**Return values****Remarks**

A field must be created with the CreateField method. Once added, however, the Field will be destroyed automatically by the control.

**Examples****See Also**

[DeleteFld](#), [CreateField](#), [InsertFld](#)

## AddRec Method

Appends a new Record to the Grid's RecordList, allocates space for the RecData and, optionally, sets the Record's data.

### Usage

#### C

```
iRows = (int)SendMessage(hWnd, HGM_ADDREC, 0, lpRecData);
```

#### C++

```
iRows = [CHGridObj.]AddRec([lpRecData = NULL])
```

#### VBX

```
[form.][control.]Records[ = iRecords]
```

### Arguments/Parameters

void FAR \*lpRecData

A pointer to a filled record data structure for initializing the new record or NULL

int iRecords

The desired number of rows in the grid; rows will be appended or deleted as necessary

### Return values

int iRows

The number of rows in the Grid

### Remarks

After adding records to the Grid, it is necessary to call the Update method to properly update the display and scroll ranges.

Adding records in Visual Basic only add a blank record, which can then be initialized by setting the Data property.

### See Also

DeleteRec, InsertRec

## ColToFld Method

Translates from Column index to Field index, which can be different if a ColMap is used

### Usage

#### C

```
iField = HGColToFld(hWnd, iCol);
```

#### C++

```
iField = [CHGridObj.]ColToFld(iCol);
```

#### VBX

```
iField = VGColToFld([control.]hWnd, iCol);
```

### Arguments/Parameters

<code>int iCol</code>	The column index to be converted
-----------------------	----------------------------------

### Return values

<code>int iField</code>	The converted field index
-------------------------	---------------------------

### Remarks

Column indices are positions in the displayed grid; Field indices are positions in the Grid data structure. Hidden columns retain a column index indicating the position they at which they would appear if shown.

### See Also

[ColMap](#), [FldToCol](#), [RecToRow](#), [RowToRec](#)

## CreateField Method

Creates a new Field dynamically, which must be added or inserted into the Grid

### Usage

#### C

```
hFld = HGFieldCreate(lpName, wState, iColWidth, iDropHeight, cDataClass, cDataType, cCodeClass, cCodeType, iSize, cCtlType, lCtlStyle, lpFmt);
```

#### C++

```
[CHGGridObj.]FieldCreate(lpName, wState, iColWidth, iDropHeight, cDataClass, cDataType, cCodeClass, cCodeType, iSize, cCtlType, lCtlStyle, lpFmt);
```

#### VBX

```
hFld = VGFieldCreate(lpName, wState, iColWidth, iDropHeight, iDataClass, iDataType, iCodeClass, iCodeType, iSize, iCtlType, lCtlStyle, lpFmt)
```

### Arguments/Parameters

LPSTR lpName	The <u>FldName</u> of the field to be created
WORD wState	The <u>FldState</u> of the field to be created
int iColWidth	The <u>ColWidth</u> of the field to be created
int iDropHeight	The <u>DropHeight</u> of the field if it is a DropDown or ComboBox field
char cDataClass	One of the <u>data class character codes</u>
char cDataType	One of the <u>data type character codes</u>
char cCodeClass	One of the <u>data class character codes</u> (ListBox and DropDown fields only)
char cCodeType	One of the <u>data type character codes</u> (ListBox and DropDown fields only)
int iSize	The size in bytes of the Data stored in the grid for each cell in this Field. This parameter is only used for string types; for other types, the size is predefined. For list Fields that use Codes to determine the list selection, the iSize is the size of each Code item.
char cCtlType	One of the <u>control type constants</u>
LONG lCtlStyle	A collection of valid style flags for the fields control type.
LPSTR lpFmt	A <u>FldFormat</u> for the field to be created

### Return values

HFLD hFld	A 32-bit handle to a field data structure or NULL if the field could not be created
-----------	---

### Remarks

Creating fields dynamically is used when field attributes can only be determined at run time. If the programmer wishes to conditionally display one or more of a fixed

set of fields with known attributes, it may be easier to specify all of the fields in the Grid Resource and hide the ones that are not to be displayed.

Dynamically created fields must be explicitly added or inserted into the Grid. They must also be destroyed by the programmer using the DestroyField method.

## DeleteFld Method

Deletes a Field from the Grid

### Usage

#### C

```
bSuccess = (BOOL)SendMessage(hWnd, HGM_DELETEFLD, iCol, 0L);
```

#### C++

```
bSuccess = [CHGridObj.]DeleteFld(iCol);
```

#### VBX

```
bSuccess = VGDeleteFld([control.]hWnd, iCol)
```

### Arguments/Parameters

<code>int iCol</code>	The column index of the field to delete
-----------------------	---

### Return values

<code>BOOL bSuccess</code>	TRUE if the field was deleted successfully
----------------------------	--

### See Also

[AddFld](#), [InsertFld](#)

## DeleteRec Method

Removes a record from the Grid

### Usage

#### C

```
bSuccess = (BOOL)SendMessage(hWnd, HGM_DELETEREC, (WPARAM)iRow, 0L);
```

#### C++

```
bSuccess = [CHGridObj.]DeleteRec(iRow);
```

#### VBX

```
bSuccess = VGDeleteRec([control.]hWnd, iRow)
```

### Arguments/Parameters

<code>int iRow</code>	The row index of the record to be deleted
-----------------------	---

### Return values

<code>BOOL bSuccess</code>	TRUE if the record was deleted successfully
----------------------------	---

### Remarks

After removing records from the Grid, it is necessary to call the Update method to properly update the display and scroll ranges.

### See Also

[AddRec](#), [InsertRec](#)



## DestroyField Method

Destroys a field created with the [CreateField](#) method

### Usage

#### C

```
bSuccess = HGFieldDestroy(hFld);
```

#### C++

```
bSuccess = FieldDestroy(hFld);
```

#### VBX

```
bSuccess = VGFieldDestroy(hFld)
```

### Arguments/Parameters

HFLD hFld

The 32-bit handle of a field structure returned by the CreateField method

### Return values

BOOL bSuccess

TRUE if the field has been destroyed successfully

### Remarks

Programmers should only destroy fields that are

1) created dynamically using the CreateField method.

*and*

2) not present in the Grid when it is destroyed.

Field's that are specified in the Grid Resource are created automatically when the Grid is initialized, these fields will be destroyed automatically when the Grid is destroyed. Fields created with CreateField that are inserted or added into the grid will also be automatically destroyed when the Grid is destroyed.

### See Also

[CreateField](#)

## FindFld Method

Gets the column position of a field given the FldName

### Usage

#### C

```
iCol = (int)SendMessage(hWnd, HGM_FINDFLD, 0, (DWORD)lpFldName);
```

#### C++

```
iCol = FindFld(lpFldName);
```

#### VBX

```
iCol = VGFindFld([control.]hWnd, lpFldName)
```

### Arguments/Parameters

LPSTR lpFldName

The FldName text string of the field to be located

### Return values

int iCol

The column index of the field or  
HGERR\_NOTFOUND

## FldToCol Method

Translates from Field index to Column index, which can be different if a ColMap is used

### Usage

#### C

```
iCol = HGFldToCol(hWnd, iField);
```

#### C++

```
iCol = [CHGridObj.]FldToCol(iField);
```

#### VBX

```
iCol = VGFldToCol([control.]hWnd, iField)
```

### Arguments/Parameters

<code>int iField</code>	The column index to be converted
-------------------------	----------------------------------

### Return values

<code>int iCol</code>	The converted field index
-----------------------	---------------------------

### Remarks

Column indices are positions in the displayed grid; Field indices are positions in the Grid data structure. Hidden columns retain a column index indicating the position they at which they would appear if shown.

### See Also

[ColMap](#), [ColToFld](#), [RecToRow](#), [RowToRec](#)

## GetCurRec Method

Retrieves the RecData of the currently selected record

### Usage

#### C

```
bSuccess = (BOOL)SendMessage(hWnd, HGM_GETCURREC, 0, lpRecData);
```

#### C++

```
bSuccess = [CHGridObj.]GetCurRec(lpRecData);
```

#### VBX

Not Used

### Arguments/Parameters

void FAR \*lpRecData

A pointer to a record data structure to which the records data will be copied

### Return values

BOOL bSuccess

TRUE if the RecData was retrieved successfully

## InsertFld Method

Inserts a field at a given position in the Grid

### Usage

#### C

```
iFld = (int)SendMessage(hWnd, HGM_INSERTFLD, iAtIndex, hFld);
```

#### C++

```
iFld = [CHGridObj.]InsertFld(iAtIndex, hFld);
```

#### VBX

```
iFld = VGInsertFld([control.]hWnd, iAtIndex, hFld)
```

### Arguments/Parameters

<code>int iAtIndex</code>	The field index at which to insert the field
<code>HFLD hFld</code>	The 32-bit handle of a field structure returned by the CreateField method

### Return values

<code>int iFld</code>	The column index of the inserted field
-----------------------	--

### Remarks

Programmers using the [DragCols](#) attribute should be aware of the differences between field index and column index when inserting fields, as discussed in the [ColMap](#) attribute section. The field is inserted at a column index equal to `iAtIndex` and must be moved to a desired column position using the [MoveCol](#) method.

## InsertRec Method

Inserts a new Record into the Grid's RecordList, allocates space for the RecData and, optionally, sets the RecData

### Usage

#### C

```
iRow = (int)SendMessage(hWnd, HGM_INSERTREC, (WPARAM)iAtRow,  
(LPARAM)lpRecData);
```

#### C++

```
iRow = [CHGridObj.]InsertRec(iAtRow[, lpRecData = NULL]);
```

#### VBX

```
iRow = VGInsertRec([control.]hWnd, iAtRow)
```

### Arguments/Parameters

int iAtRow

The row index at which to insert a record

void FAR \*lpRecData

A pointer to a filled record data structure for initializing the new record or NULL

### Return values

int iRow

The row index of the inserted record

### Remarks

After adding records to the Grid, it is necessary to call the Update method to properly update the display and scroll ranges.

The Visual Basic method, HGInsertRec, can only insert a blank record that can then be initialized by setting the Data property.

### See Also

AddRec, DeleteRec

## Invalidate Method

Invalidates individual cells or a range of cells in the Grid

### Usage

#### C

```
bSuccess = HGridInvalidateCell(hWnd, iCol1, iRow1);  
bSuccess = HGridInvalidateRange(hWnd, iCol1, iRow1, iCol2, iRow2);
```

#### C++

```
bSuccess = [CHGridObj.]InvalidateCell (iCol1, iRow1);  
bSuccess = [CHGridObj.]InvalidateRange(iCol1, iRow1, iCol2, iRow2);
```

#### VBX

```
bSuccess = VGInvalidateCell ([control.]hWnd, iCol1, iRow1)  
bSuccess = VGInvalidateRange(hWnd, iCol1, iRow1, iCol2, iRow2)
```

### Arguments/Parameters

`int iCol1, iRow1, iCol2, iRow2` Specify a cell's coordinates or a range of cells bordered by two sets of coordinates

### Return values

`BOOL bSuccess` TRUE if the region was invalidated correctly

### Remarks

The invalidated region is repainted when the Grid receives a WM\_PAINT message.

## MoveCol Method

Moves a column to a different position in the Grid

### Usage

#### C

```
iNewIndex = (int)SendMessage(hWnd, HGM_MOVECOL, (WPARAM)iFrom,  
(LPARAM)iTo);
```

#### C++

```
iNewIndex = [CHGridObj.]MoveCol(iFrom, iTo);
```

#### VBX

```
iNewIndex = VGMoveCol([control.]hWnd, iFrom, iTo)
```

### Arguments/Parameters

<code>int iFrom</code>	The column index of the field to be moved
<code>int iTo</code>	The desired destination column index

### Return values

<code>int iNewIndex</code>	The actual column index of the moved field
----------------------------	--

### Remarks

If the DragCols attribute is set, moving a field only affects its column position, not its data's position in the record data structure. Otherwise, the data is move within the record to match the displayed field order.



## MoveRow Method

Moves a record to a different row in the Grid

### Usage

#### C

```
iRow = (int)SendMessage(hWnd, HGM_MOVEROW, (WPARAM)iFrom, (LPARAM)iTo);
```

#### C++

```
iRow = [CHGridObj.]MoveRow(iFrom, iTo);
```

#### VBX

```
iRow = VGMoveRow([control.]hWnd, iFrom, iTo)
```

### Arguments/Parameters

<code>int iFrom</code>	The row index of the record to be moved
<code>int iTo</code>	The desired destination row index

### Return values

<code>iRow</code>	The row index of the moved record
-------------------	-----------------------------------

### Remarks

Other records will be shifted either up or down by this action depending on the relationship between the original and new record positions.

## OffsetPtr Method

Obtains a pointer into the record data structure for a field in a record data structure, given a column index

### Usage

#### C

```
lpCellData = HGOffsetPtr(hWnd, iCol, lpRecData)
```

#### C++

```
lpCellData = [CHGridObj.]OffsetPtr(iCol, lpRecData)
```

#### VBX

Not Used

### Arguments/Parameters

`int iCol`

The column index of the field

`void FAR *lpRecData`

A pointer to a record data structure

### Return values

`void FAR *lpCellData`

A pointer to a field within the record data structure

### See Also

[ColToFld](#), [RecData](#)

## RecToRow Method

Returns a Row index given a record number

### Usage

#### C

```
iRow = HGRectoRow(hWnd, lRecNum);
```

#### C++

```
iRow = [CHGridObj.]RectoRow(lRecNum);
```

#### VBX

```
iRow = VGRectoRow([control.]hWnd, lRecNum)
```

### Arguments/Parameters

long lRecNum

The record number to be converted

### Return values

int iRow

The converted row index or -1 if the record is not currently in the buffer.

### Remarks

Row indices are row positions in the displayed grid, ranging from 0 to BufferSize if buffering is used. Record numbers are positions in the entire virtual record space. They range from 0 to MaxRec.

### See Also

ColMap, ColToFld, FldToCol, RowToRec

## Reset Method

Empties the Grid of all RecData

### Usage

#### C

```
SendMessage(hWnd, HGM_RESETCONTENT, (WPARAM)bRedraw, 0L);
```

#### C++

```
[CHGridObj.]ResetContent([bRedraw = TRUE]);
```

#### VBX

```
[control.]Clear
```

### Arguments/Parameters

BOOL bRedraw

TRUE forces an immediate update

### Remarks

The field/column structure of the Grid is preserved by this operaton.

## RowToRec Method

Returns a record number given a Row Index

### Usage

#### C

```
lRecNum = HGRowToRec(hWnd, iRow);
```

#### C++

```
lRecNum = [CHGridObj.]RowToRec(iRow);
```

#### VBX

```
lRecNum = VGRowToRec([control.]hWnd, iRow)
```

### Arguments/Parameters

<code>int iRow</code>	The converted row index or -1 if the record is not currently in the buffer.
-----------------------	---

### Return values

<code>long lRecNum</code>	The record number to be converted
---------------------------	-----------------------------------

### Remarks

Row indices are row positions in the displayed grid, ranging from 0 to BufferSize if buffering is used. Record numbers are positions in the entire virtual record space. They range from 0 to MaxRec.

### See Also

ColMap, ColToFld, FldToCol, RecToRow

## Update Method

Forces the Grid to redraw itself based on changed data, updating scrollbars, column widths and row height

### Usage

#### C

```
SendMessage (hWnd, HGM_UPDATE, (WPARAM)bErase, 0L);
```

#### C++

```
[CHGridObj.]Update (bErase);
```

#### VBX

```
[control.]Refresh
```

### Arguments/Parameters

BOOL bErase

TRUE causes the Grid to erase its background before repainting.

### Remarks

In the experience of our technical support staff, failure to update a grid is one of the most common sources of confusion. When adding records to the grid using the [AddRec](#) method, for example, a vertical scrollbar will not automatically appear when there are more records than the Grid can display. Calling Update in this case will force the Grid to recalculate its display settings and create the scroll bar if it is needed. In general, whenever the Grid's structure is modified through code (as opposed to user interaction), the Update method should be called to ensure that the displayed Grid accurately reflects the changes. The Grid does *not* update automatically when modified through code, which allows the programmer to make many modifications and have them take effect in a single repainting of the Grid.

## HGrid Events

<u>Bottom</u>	<u>KillFocus</u>	<u>SelChanging</u>
<u>ColMoved</u>	<u>RecChanged</u>	<u>SelExtending</u>
<u>ColSized</u>	<u>RecDelete</u>	<u>SetFocus</u>
<u>Destroy</u>	<u>RecNew</u>	<u>SpaceError</u>
<u>DoubleClick</u>	<u>RecSwitch</u>	<u>Top</u>
<u>HScroll</u>	<u>RowSized</u>	<u>VScroll</u>
<u>Initialize</u>	<u>SelChange</u>	

## Bottom Event

The user has attempted to scroll off the bottom of the table.

### Usage

#### C/C++

Notification code: **HGN\_BOTTOM**

#### VBX

**Sub** *ctlname*\_Bottom()

## ColMoved Event

The user has dragged a column in the Grid to a new location

### Usage

#### C/C++

Notification code: **HGN\_COLMOVED**

#### VBX

```
Sub ctlname_ColMoved()
```



## ColSized Event

The user has resized a column in the Grid

### Usage

#### C/C++

Notification code: **HGN\_COLSIZED**

#### VBX

```
Sub ctlname_ColSized()
```

## Destroy Event

The control is about to be destroyed

### Usage

#### C/C++

Notification code: **HGN\_DESTROY**

#### VBX

```
Sub ctlname_Destroy()
```

## DoubleClick Event

The user has double-clicked on an cell

### Usage

#### C/C++

Notification code: **HGN\_DBLCLK**

#### VBX

```
Sub ctlname_DblClick()
```

## HScroll Event

The user has scrolled the grid horizontally

### Usage

#### C/C++

Notification code: **HGN\_HSCROLL**

#### VBX

```
Sub ctlname_HScroll()
```

## Initialize Event

Occurs just prior to the control loading data from the DataSource, if one is specified. Allows fields to be initialized before the data is loaded.

### Usage

#### C/C++

Not used.

#### VBX

```
Sub ctlname_Initialize()
```

## KillFocus Event

The control has lost input focus

### Usage

#### C/C++

Notification code: **HGN\_KILLFOCUS**

#### VBX

```
Sub ctlname_KillFocus()
```

## RecChanged Event

The contents of a record have changed

### Usage

#### C/C++

Notification\_code: **HGN\_RECCHANGED**

#### VBX

```
Sub ctlname_RecChanged()
```

### Remarks

The SelectionExtent gives the row index of the record

## RecDelete Event

A record is about to be deleted

### Usage

#### C/C++

Notification\_code: **HGN\_RECDELETE**

#### VBX

**Sub** *ctlname\_RecDelete* ()

### Remarks

The SelectionExtent gives the row index of the record to be deleted



## RecNew Event

A new record was inserted

### Usage

#### C/C++

Notification\_code: HGN\_RECNEW

#### VBX

Sub ctlname\_RecNew()

### Remarks

The SelectionExtent gives the row index of the new record

## RecSwitch Event

The selection has been moved to a different record.

### Usage

#### C/C++

Notification code: HGN\_REC SWITCH

#### VBX

```
Sub ctlname_RecSwitch()
```

### Remarks

The SelectionExtent gives the row index of the new record

## RowSized Event

The user has resized the rows of the Grid

### Usage

#### C/C++

Notification code: **HGN\_ROWSIZED**

#### VBX

```
Sub ctlname_RowSized()
```

## SelChange Event

The Selection has changed.

### Usage

#### C/C++

Notification code: **HGN\_SELCHANGE**

#### VBX

```
Sub ctlname_SelChange()
```

## SelChanging Event

The user is attempting to move the input focus to another cell location.

### Usage

#### C/C++

Notification code: **HGN\_SELCHANGING**

#### VBX

```
Sub ctlname_SelChanging()
```

### Remarks

The Grid sends this notification *before* the selection is actually changed. The Marker attribute contains the row and column index of the desired new cell location. The application can prevent the selection from moving by resetting the Marker while processing this notification. The SelectionExtent gives the location of the current selection.

### See Also

SelExtending

## SelExtending Event

The user is attempting to extend the selection.

### Usage

#### C/C++

Notification code: HGN\_SELEXTENDING

#### VBX

```
Sub ctlname_SelExtending()
```

### Remarks

The Grid sends this notification *before* the selection is actually changed. The Marker attribute contains the row and column index of the desired new selection extent. The application can prevent the selection from moving by resetting the Marker while processing this notification. The SelectionExtent gives the location of the current selection.

### See Also

SelChanging

## SetFocus Event

The control has gained the input focus.

### Usage

#### C/C++

Notification code: **HGN\_SETFOCUS**

#### VBX

**Sub** *ctlname\_SetFocus* ()

## SpaceError Event

The control is unable to perform an operation because of memory constraints

### Usage

#### C/C++

Notification code: **HGN\_ERRSPACE**

#### VBX

```
Sub ctlname_ErrSpace()
```



## Top Event

The user has attempted to scroll off the top of the table.

### Usage

#### C/C++

Notification code: **HGN\_TOP**

#### VBX

**Sub** *ctlname\_Top*()

## VScroll Event

The user has scrolled the grid vertically

### Usage

#### C/C++

Notification code: **HGN\_VSCROLL**

#### VBX

**Sub** *ctlname\_VScroll* ()

## Control Type Constants

<b><u>Control Constant</u></b>	<b>Type of Control</b>
HGCTL_COMBO	Combo Box
HGCTL_EDIT	Edit Control
HGCTL_LIST	List Box
HGCTL_DROP	Drop-down List Box
HGCTL_CHECK	Check Box

## HGrid Field State Flags

<b><u>Flag Constant</u></b>	<b>Meaning of TRUE value</b>
HGFF_HIDDEN	The field is not visible
HGFF_BROWSE	The field is not editable
HGFF_USECODE	The cell Data corresponds

## HGrid State Flags

<b><u>Flag Constant</u></b>	<b>Meaning of TRUE value</b>
HGF_CHANGED	The contents of the control have been changed

## HGrid Record State Flags

<b><u>Flag Constant</u></b>	<b>Meaning of TRUE value</b>
HGRF_CHANGED	The record was edited
HGRF_NEW	The record is user-added

## HGrid Window Text

This sample window text string is expanded below to show the meaning of each component:

`MyGrid`

`MyGrid` The name of the Grid resource (of the type RT\_HGRID) from which to construct the control, or blank to start with an empty Grid.

<b><u>Field Type</u></b>	<b>Cast CWnd * to</b>
CheckBox	CHBCheck *
Combo Box	CHComb *
Drop-Down List	CHComb *
Edit Control	CHEdit *
List Box	CHList *

## HList, The WinWidgets ListBox

- ▣ Attributes
- ▣ Methods
- ▣ Events

HList is a listbox control that displays lists of any type of data supported by the DataEngine. Each item in the list may also have an associated Code, which is not displayed but can be used to select or sort items in the list. Codes are also a convenient place to store a pointer to additional item information.

A complete set of methods makes appending, inserting, deleting, selecting and retrieving items from the list as easy as possible.

HList supports single, multiple and extended selection modes, multiple-column and tab-expanded display, and sorting by Data or Codes. HList has standard and 3D border styles and the ability to highlight itself upon gaining focus, helping users track their position on forms.

### Additional Topics

- ▣ Hot-Linking the ListBox to your data
- ▣ Using HList with the Visual Basic Data Control
- ▣ Using Codes in the ListBox

## HList Attributes

<u>Background</u>	<u>DataLink</u>	<u>SelCount</u>
<u>BorderStyle</u>	<u>DataSize</u>	<u>Selection</u>
<u>Changed</u>	<u>DataType</u>	<u>SelectionMode</u>
<u>Code</u>	<u>Font</u>	<u>SelectMode</u>
<u>CodeClass</u>	<u>Format</u>	<u>SellItems</u>
<u>CodeLink</u>	<u>HiliteBrush</u>	<u>SortMode</u>
<u>CodeSize</u>	<u>HiliteOnFocus</u>	<u>State</u>
<u>CodeType</u>	<u>Hunger</u>	<u>TabStops</u>
<u>ColWidth</u>	<u>MultiCol</u>	<u>Text</u>
<u>Count</u>	<u>NonIntHeight</u>	<u>TextColor</u>
<u>Data</u>	<u>Quiet</u>	<u>TextLen</u>
<u>DataClass</u>	<u>RedrawMode</u>	<u>TopIndex</u>

## Background Attribute

The color or pattern used to paint the background of the edit control

### Usage

#### C

```
hbrBkgnd = (HBRUSH) SendMessage(hWnd, HLM_GETBKGND BRUSH, 0, 0L);  
hbrOldBkgnd = (HBRUSH) SendMessage(hWnd, HLM_SETBKGND BRUSH,  
(WPARAM) hbrNewBrush, 0L);
```

#### C++

##### OWL

```
hbrBkgnd = [THListObj.]GetBkgndBrush(void);  
hbrOldBkgnd = [THListObj.]SetBkgndBrush(hbrNewBrush);
```

##### MFC

```
pBkgnd = [CHListObj.]GetBkgndBrush(void);  
pOldBkgnd = [CHListObj.]SetBkgndBrush(pNewBrush);
```

#### VBX

```
[form.][control.]BackColor[ = color ]
```

See Visual Basic Language Reference, "BackColor, ForeColor Properties"

### Arguments/Parameters

HBRUSH hbrNewBrush

Handle of a new brush to be set as the background brush

CBrush pNewBrush

Pointer to a CBrush object containing the new background brush handle

### Return values

HBRUSH	hbrBkgnd	Handle to the current background brush
HBRUSH	hbrOldBkgnd	Handle to the previous background brush
CBrush	*pBkgnd	Pointer to a CBrush object containing the current background brush handle
CBrush	*pOldBkgnd	Pointer to a CBrush object containing the previous background brush handle

### **Remarks**

C and C++ applications are responsible for destroying any brushes they create.



## BorderStyle Attribute

HList supports four different border styles: none, standard, indented and extruded.

### Usage

#### C/C++

Window Styles:

HLS\_BORDER3D

HLS\_INDENT

HLS\_EXTRUDE

#### VBX

[*form.*][*control.*]**BorderStyle**[= *None/Standard/Indented/Bump*]

### Remarks

The BorderStyle attribute can only be set at design time.

## Changed Attribute

A boolean value indicating if the Data has been changed since it was last set

### Usage

#### C

```
bChanged = (BOOL)SendMessage(hWnd, HLM_HASCHANGED, 0, 0L);  
SendMessage(hWnd, HLM_SETCANGED, bVal, 0L);
```

#### C++

```
bChanged = [CHListObj.]HasChanged();  
[CHListObj.]SetChanged(bVal);
```

#### VBX

Not Used

### Arguments/Parameters

BOOL bVal

The new value for the Changed attribute

### Return Value

BOOL bChanged

TRUE if the Data has been changed since it was last set

### See Also

[Change event](#)

## Code Attribute

The native (binary) data maintained but not displayed by the control for each item in the List

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HLM_GETCODE, (WPARAM) iIndex,  
    (LPARAM) lpCode);
```

```
bSuccess = HLGetCode(hWnd, iIndex, lpCode);
```

```
bSuccess = SendMessage(hWnd, HLM_SETCODE, (WPARAM) iIndex,  
    (LPARAM) lpCode);
```

```
bSuccess = HLSetCode(hWnd, iIndex, lpCode);
```

Get the code of item for the current [Selection](#)

```
iBytesCopied = (int)SendMessage(hWnd, HLM_GETCURCODE, wSize,  
    (LPARAM) lpBuf);
```

#### C++

```
bSuccess = [CHListObj.]GetCode(iIndex, lpCode);
```

```
bSuccess = [CHListObj.]SetCode(iIndex, lpCode);
```

Get the code of item for the current [Selection](#)

```
iBytesCopied = [CHListObj.]GetCurCode(lpBuf [, wSize = -1]);
```

#### VBX

```
[form.][control.]Code(iIndex) [= stringexpression]
```

The **Code** attribute is a string array; *iIndex* is a required parameter

### Arguments/Parameters

int iIndex	The index of the item
void FAR *lpCode	Pointer to a buffer for the code
WORD wSize	Maximum number of bytes to copy (used only for strings).

### Return values

BOOL bSuccess	TRUE if the operation is a success
int iBytesCopied	Number of bytes copied.

### Remarks

The Code attribute for an item cannot be set for a list that is sorted by codes.

### See Also

[CodeClass](#), [CodeLink](#), [CodeSize](#), [CodeType](#), [Data](#)

## CodeClass Attribute

One of the data classes defined in the [Data Engine](#) chapter

### Usage

#### C

```
cCodeClass = (char)SendMessage(hWnd, HLM_GETCODECLASS, 0, 0L);
```

#### C++

```
cCodeClass = [CHListObj.]GetCodeClass();
```

#### VBX

```
[form.][control.]CodeClass
```

### Return values

**char** cCodeClass                      One of the [data class character codes](#)

### Remarks

The CodeClass attribute can only be set at design time. Control over this attribute is provided by the control design dialogs in Dialog Editor and Resource Workshop, and by the Properties dialog in Visual Basic. When creating windows dynamically in C and C++, the DataClass is included in the [WindowText](#).

### See Also

[Code](#), [CodeType](#)

## CodeLink Attribute

A pointer to the variable or buffer that will be updated with the new item's code when the Selection is changed

### Usage

#### C

```
lpCodeLink = (void FAR *)SendMessage(hWnd, HLM_GETCODELINK, 0, 0L);  
lBytesCopied = (LONG)SendMessage(hWnd, HLM_SETCODELINK, (WPARAM)bSelect,  
(LPARAM)(LPVOID)lpNewLink);  
HLSetCodeLink(hWnd, lpCode, bSelect);
```

#### C++

```
lpCodeLink = [CHListObj.]GetCodeLink();  
bSuccess = [CHListObj.]SetCodeLink(lpNewLink [, bSelect = TRUE]);
```

#### VBX

Not Used

### Arguments/Parameters

void FAR \*lpNewLink  
BOOL bSelect

Pointer to program data  
If TRUE, selection will be set to match the contents of lpNewLink

### Return values

void FAR \*lpCodeLink  
  
BOOL bSuccess  
LONG lBytesCopied

Pointer to the current CodeLink. NULL if no CodeLink has been set.  
TRUE if CodeLink was set successfully  
Number of bytes copied.

### See Also

[Code](#)

## CodeSize Attribute

The size of each item's Code in bytes

### Usage

#### C

```
iCodeSize = SendMessage(hWnd, HLM_GETCODESIZE, (WPARAM)iIndex, 0L);
```

#### C++

```
iCodeSize = [CHListObj.]GetCodeSize([iIndex = -1]);
```

#### VBX

Not Used

### Arguments/Parameters

`int iIndex`

The index of an item--only necessary with character string CodeTypes.

### Return values

`int iCodeSize`

The size of the Code

### Remarks

The CodeSize attribute may be variable only for NULL-terminated strings.

### See Also

[CodeType](#)

## CodeType Attribute

One of the data types defined in the [DataEngine](#) chapter

### Usage

#### **C**

```
cCodeType = (char)SendMessage(hWnd, HLM_GETCODETYPE, 0, 0L);
```

#### **C++**

```
cCodeType = [CHListObj.]GetCodeType();
```

#### **VBX**

```
[form.][control.]CodeType
```

### Return values

char cCodeType

One of the [data type character codes](#)

### See Also

[Code](#), [CodeClass](#), [CodeSize](#)

## ColWidth Attribute

The width of columns in a MultiColumn listbox

### Usage

#### C

```
bSuccess = (BOOL)SendMessage(hWnd, HLM_SETCOLUMNWIDTH, (WPARAM)iWidth,  
0L);
```

#### C++

```
bSuccess = [CHListObj.]SetColumnWidth(iWidth);
```

#### VBX

```
[form.][control.]ColumnWidth[= integer]
```

### Arguments/Parameters

int iWidth

The new ColumnWidth value

### Return values

BOOL bSuccess

TRUE if the ColumnWidth was set successfully



## Count Attribute

The number of items in the List

### Usage

#### C

```
iCount = SendMessage(hWnd, HLM_GETCOUNT, 0, 0L);
```

#### C++

```
iCount = [CHListObj.]GetCount();
```

#### VBX

```
[form.][control.]Count
```

### Return values

int iCount

The current number of items in the List

## Data Attribute

The native (binary) data maintained and displayed by the control for each item in the List

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HLM_GETDATA, (WPARAM) iIndex,  
    (LPARAM) lpData);  
bSuccess = HLGetData(hWnd, iIndex, lpData);  
Get the data of item at CursorPosition:  
iBytesCopied = (int)SendMessage(hWnd, HLM_GETCURDATA, wSize,  
    (LPARAM) lpBuf);
```

#### C++

```
bSuccess = [CHListObj.]GetData(iIndex, lpData);  
Get the data of item at CursorPosition  
iBytesCopied = [CHListObj.]GetCurData(lpBuf [, wSize = -1]);
```

#### VBX

[*form.*][*control.*]**Data**(*iIndex*)

The **Data** attribute is a string array; *iIndex* is a required parameter

### Arguments/Parameters

int iIndex	The index of the item
void FAR *lpData	Pointer to a buffer for the Data
WORD wSize	Maximum number of bytes to copy (used only for strings).

### Return values

BOOL bSuccess	TRUE if the operation is a success
int iBytesCopied	Number of bytes copied.

### See Also

[Code](#), [DataClass](#), [DataLink](#), [DataSize](#), [DataType](#)

## DataClass Attribute

One of the data classes defined in the [Data Engine](#) chapter

### Usage

#### C

```
cDataClass = (char)SendMessage(hWnd, HLM_GETDATACLASS, 0, 0L);
```

#### C++

```
cDataClass = [CHListObj.]GetDataClass();
```

#### VBX

```
[form.][control.]DataClass
```

### Return values

char cDataClass                      One of the [data class character codes](#)

### Remarks

The DataClass attribute can only be set at design time. Control over this attribute is provided by the control design dialogs in Dialog Editor and Resource Workshop, and by the Properties dialog in Visual Basic. When creating windows dynamically in C and C++, the DataClass is included in the [WindowText](#).

### See Also

[Data](#), [DataType](#)

## DataLink Attribute

A pointer to the variable or buffer that will be updated with the new item's Data when the CursorPosition is changed

### Usage

#### C

```
lpDataLink = (void FAR *)SendMessage(hWnd, HLM_GETDATALINK, 0, 0L);  
lBytesCopied = (LONG)SendMessage(hWnd, HLM_SETDATALINK, (WPARAM)bSelect,  
(LPARAM)(LPVOID)lpNewLink);  
HLSetDataLink(hWnd, lpData, bSelect);
```

#### C++

```
lpDataLink = [CHListObj.]GetDataLink();  
bSuccess = [CHListObj.]SetDataLink(lpNewLink [, bSelect = TRUE]);
```

#### VBX

Not Used

### Arguments/Parameters

void FAR *lpNewLink	Pointer to program data
BOOL bSelect	If TRUE, selection will be set to match the contents of lpNewLink

### Return values

void FAR *lpDataLink	Pointer to the current DataLink. NULL if no DataLink has been set.
BOOL bSuccess	TRUE if DataLink was set successfully
LONG lBytesCopied	Number of bytes copied.

### See Also

[Data](#)

## DataSize Attribute

The size of each item's Data in bytes

### Usage

#### C

```
iDataSize = SendMessage(hWnd, HLM_GETDATASIZE, (WPARAM)iIndex, 0L);
```

#### C++

```
iDataSize = [CHListObj.]GetDataSize([iIndex = -1]);
```

#### VBX

Not Used

### Arguments/Parameters

<code>int iIndex</code>	The index of an item, only necessary with character string DataTypes.
-------------------------	---

### Return values

<code>int iDataSize</code>	The size of the Data
----------------------------	----------------------

### Remarks

The DataSize attribute may be variable only for NULL-terminated strings.

### See Also

[DataType](#)

## DataType Attribute

One of the data types defined in the [DataEngine](#) chapter

### Usage

#### C

```
cDataType = (char)SendMessage(hWnd, HLM_GETDATATYPE, 0, 0L);
```

#### C++

```
cDataType = [CHListObj.]GetDataType();
```

#### VBX

```
[form.][control.]DataType
```

### Return values

char cDataType

One of the [data type character codes](#)

### See Also

[Data](#), [DataClass](#), [DataSize](#)

## Font Attribute

The font used by the control

### Usage

#### C

```
hfFont = (HFONT)SendMessage(hWnd, HLM_GETFONT, 0, 0L);  
hfOldFont = SendMessage(hWnd, HLM_SETFONT, (WPARAM)hfNewFont,  
(LPARAM)bRedraw);
```

#### C++

##### OWL

```
hfFont = [THListObj.]GetFont();  
hfOldFont[THListObj.]SetFont(hfNewFont [, bRedraw = TRUE] );
```

##### MFC

```
pFont = [CHListObj.]GetFont();  
pOldFont[CHListObj.]SetFont(pNewFont [, bRedraw = TRUE] );
```

#### VBX

```
[form.][control.]FontBold[= boolean]  
[form.][control.]FontItalic[= boolean]  
[form.][control.]FontName[= font]  
[form.][control.]FontSize[= points]  
[form.][control.]FontStrikethru[= boolean]  
[form.][control.]FontUnderline[= boolean]
```

See Visual Basic Language Reference, "FontName Property"

### Arguments/Parameters

HFONT hfNewFont	Handle of the font to be set
CFont *pFont	Pointer to a CFont object containing the handle to the font to be set
BOOL bRedraw	A value of TRUE causes the control to repaint immediately

### Return values

HFONT hfFont	Handle to the control's current font
HFONT hfOldFont	Handle to the control's previous font
CFont *hfFont	Pointer to a CFont object containing the handle to the control's current font
CFont *fOldFont	Pointer to a CFont object containing the handle to the control's previous font

### Remarks

C and C++ applications are responsible for destroying any fonts they create.

## Format Attribute

A NULL-terminated character string that describes the way the Data is to be displayed

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HLM_GETFORMAT, (WPARAM)iMaxBytes,  
(LPARAM)lpstrBuf);  
lBytesCopied = SendMessage(hWnd, HLM_SETFORMAT, (WPARAM)bRedraw,  
(LPARAM)lpstrBuf);
```

#### C++

```
lBytesCopied = [CHListObj].GetFormat(lpstrBuf [, iMaxBytes = -1] );  
lBytesCopied = [CHListObj].SetFormat(lpstrBuf [, bRedraw = TRUE] );
```

#### VBX

```
[form].[control].FormatString[= string]
```

### Arguments/Parameters

LPSTR lpstrBuf	Buffer that contains a new format string or will receive the existing one
int iMaxBytes	The maximum bytes to copy to the buffer. A value of -1 copies the entire format string.
BOOL bRedraw	A value of TRUE causes the control to redraw immediately.

### Return values

LONG lBytesCopied	The number of bytes actually copied to or from the buffer
-------------------	---

### Remarks

An initial format string is contained in the [WindowText](#) when a control is created.



## HiliteBrush Attribute

The color or pattern used to paint the background when the control receives focus

### Usage

#### C

```
hbrHilite = SendMessage(hWnd, HLM_GETHILITEBRUSH, 0, 0L);  
hbrOldHilite = SendMessage(hWnd, HLM_SETHILITEBRUSH,  
(WPARAM)hbrNewHilite, 0L);
```

#### C++

##### OWL

```
hbrHilite = [THListObj.]GetHiliteBrush();  
hbrOldHilite = [THListObj.]SetHiliteBrush(hbrNewHilite);
```

##### MFC

```
pHilite = [CHListObj.]GetHiliteBrush();  
pOldHilite = [CHListObj.]SetHiliteBrush(pNewHilite);
```

#### VBX

```
[form.][control.]HiliteColor[= color]
```

### Arguments/Parameters

HBRUSH hbrNewHilite	Handle of the new brush
CBrush *pNewHilite	Pointer to a CBrush object containing the handle of the new brush

### Return values

HBRUSH hbrHilite	Handle of the control's current brush
HBRUSH hbrOldHilite	Handle of the control's previous brush
CBrush *pHilite	Pointer to a CBrush object containing the handle of the current brush
CBrush *pOldHilite	Pointer to a CBrush object containing the handle of the previous brush

### Remarks

C and C++ applications are responsible for destroying any brushes they create.

## HiliteOnFocus Attribute

When set, the HiliteBrush is used to paint the background when the control receives input focus. If no HiliteBrush is selected, the control uses a white brush.

### Usage

#### C/C++

Window Style: **HLS HILITE**

#### VBX

[*form.*] [*control.*] **HiliteOnFocus**

### Remarks

The HiliteOnFocus attribute is read-only at run time

## Hunger Attribute

When set, the control swallows *Enter* and *Esc* keyboard messages and notifies its parent.

### Usage

#### C/C++

Window Style: HLS\_HUNGER

#### VBX

[*form.*] [*control.*] **Hunger**

### Remarks

The Hunger attribute is obsolete and is included here for backward compatibility. We recommend that C and C++ programmers use a Filter Procedure or Dynamic Subclassing, respectively, to implement this functionality. Hunger is read-only at run time.

## MultiCol Attribute

When set, the items will be wrapped in newspaper-style columns.

### Usage

#### C/C++

Window Style: **HLS\_MULTICOL**

#### VBX

[*form.*][*control.*]**MultiColumn**[= *True/False*]

### Remarks

The List presents a horizontal scroll bar if all items do not fit in the window. The MultiCol attribute can only be set at design time.

## NonIntHeight Attribute

When set, the control can display a partial item at the bottom of the List.

### Usage

#### C/C++

Window Style: HLS\_NONINHEIGHT

#### VBX

[*form.*][*control.*]**NonIntHeight**

### Remarks

The NonIntHeight attribute can only be set at design time.

## Quiet Attribute

When the control is in Quiet mode, it does not send notification messages to its parent. VBX controls will not fire events in Quiet mode.

### Usage

#### C

```
SendMessage(hWnd, HLM_BEQUIET, bValue, 0L);  
bQuiet = (BOOL)SendMessage(hWnd, HLM_ISQUIET, 0, 0L);
```

#### C++

```
[CHListObj.]BeQuiet(bValue);  
bQuiet = [CHListObj.]IsQuiet();
```

#### VBX

```
SendMessage(control(hWnd), HLM_BEQUIET, bValue, 0L)  
bQuiet = SendMessage(control(hWnd), HLM_ISQUIET, 0, 0L)  
See VBX Advanced Topics
```

### Arguments/Parameters

BOOL bValue

TRUE turns on Quiet mode, FALSE turns it off

### Return Value

BOOL bQuiet

TRUE if control is in Quiet Mode

## RedrawMode Attribute

When the RedrawMode is FALSE, the control will not repaint.

### Usage

#### C

```
SendMessage(hWnd, HLM_SETREDRAW, (WPARAM)bSetting, (LPARAM)bRedrawNow);
```

#### C++

```
[CHListObj.]SetRedraw(bSetting, bRedrawNow);
```

#### VBX

```
[form.][control.]RedrawMode[= boolean]
```

### Arguments/Parameters

BOOL bSetting

BOOL bRedrawNow

TRUE turn on redrawing, FALSE turns it off

If bSetting is TRUE, a TRUE value for bRedrawNow will force an immediate update.

## SelItems Attribute

An array of selected items in the List

### Usage

#### C

```
iCopied = (int)SendMessage(hWnd, HLM_GETSELITEMS, (WPARAM)iMaxItems,  
(LPARAM)lpBuf);
```

#### C++

```
iCopied = [CHListObj.]GetSelItems(lpBuf, iMaxItems);
```

#### VBX

```
[form.][control.]SelectedItems(index)
```

An array of indices of selected items. The SelectionCount property gives the size of this array.

### Arguments/Parameters

`int iMaxItems`

The maximum number of indices to copied to lpBuf

`int FAR *lpBuf`

A pointer to an array of integers

### Return values

`int iCopied`

The number of items actually copied to lpBuf

### See Also

SelectionState, SelCount



## SelCount Attribute

The number of selected items in the List

### Usage

#### C

```
iCount = (int)SendMessage(hWnd, HLM_GETSELCOUNT, 0, 0L);
```

#### C++

```
iCount = [CHListObj.]GetSelCount();
```

#### VBX

```
[CHListObj.]SelCount
```

### Return values

int iCount

The current value of SelCount

### Remarks

For lists in single selection mode, the SelCount attribute can only be 0 or 1.

### See Also

SelItems, SelectionMode

## Selection Attribute

The index of the current selection of a single-selection list or the index of the `CursorPosition` in a multiple-selection list

### Usage

#### C

```
iCurSel = (int)SendMessage(hWnd, HLM_GETCURSEL, 0, 0L);  
iResult = (int)SendMessage(hWnd, HLM_SETCURSEL, (WPARAM)iIndex, 0L);
```

#### C++

```
iCurSel = [CHListObj.]GetCurSel();  
iResult = [CHListObj.]SetCurSel(iIndex);
```

#### VBX

```
[form.][control.]Selection[= index]
```

### Arguments/Parameters

`int iIndex`

The index of the item at which to set `CursorPosition`

### Return values

`int iCurSel`  
`int iResult`

The index of the current `CursorPosition`  
HLERR\_NOTFOUND if `CursorPosition` cannot be set to the requested value

### See Also

[Code](#), [Data](#), [SelectionState](#), [Selltems](#)

## SelectionMode Attribute

The selection status (i.e. selected or unselected) for each item in the List

### Usage

#### C

```
bSelected = (BOOL)SendMessage(hWnd, HLM_GETSEL, (WPARAM)iIndex, 0L);  
bSuccess = (BOOL)SendMessage(hWnd, HLM_SETSEL, (WPARAM)iAction,  
(LPARAM)iIndex);
```

#### C++

```
bSelected = [CHListObj.]GetSel(iIndex);  
bSuccess = [CHListObj.]SetSel(iAction, iIndex);
```

#### VBX

```
[form.][control.]SelectionMode(index) [= boolean]
```

### Arguments/Parameters

<code>int iIndex</code>	The index of an item in the List
<code>int iAction</code>	One of three <a href="#">action codes</a>

### Return values

<code>bSelected</code>	TRUE if the item at <code>iIndex</code> is selected
<code>bSuccess</code>	TRUE if the action indicated by <code>iAction</code> was successful

### See Also

[SelectCode](#), [SelectData](#), [SelectItem](#), [SelectString](#)

## SelectMode Attribute

When set to *Single*, the user may select only one item at a time. When set to *Multiple* or *Extended*, the user may select more than one item. Extended selection allows the user to select blocks of items simply by dragging the mouse.

### Usage

#### C/C++

Window Styles:

HLS\_MULTISEL

HLS\_EXTENDEDSEL

#### VBX

```
[form.][control.]SelectMode[= Single/Multiple/Extended]
```

### Remarks

The SelectMode attribute can only be set at design time.

## SortMode Attribute

Determines how items in the List are to be sorted

### Usage

#### C/C++

Window Styles:

HLS\_SORTBYDATA

HLS\_SORTBYCODE

#### VBX

[*form.*][*control.*]**SortMode**[= *None/ByData/ByCodes*]

### Remarks

The SortMode attribute can only be set at design time.

## State Attribute

A collection of flags describing the state of the control

### Usage

#### C

```
wState = (WORD)SendMessage(hWnd, HLM_GETSTATE, 0, 0L);  
wNewState = (WORD)SendMessage(hWnd, HLM_SETSTATE, (WPARAM)wFlag,  
(LPARAM)bSetting);
```

#### C++

```
wState = [CHListObj.]GetState();  
wNewState = [CHListObj.]SetState(wFlag, bSetting);
```

#### VBX

Not Used

### Arguments/Parameters

WORD wFlag	One of the <u>state flags</u> to set
BOOL bSetting	New value for the selected state flag

### Return values

WORD wState	Contains the current state flag values. An individual flag value can be determined by bitwise AND'ing this value with the flag, itself.
WORD wNewState	The new state flag values after setting

## TabStops Attribute

An array of integer tabstops representing spacing in characters

### Usage

#### C

```
Window Style: HLS_USETABS  
bSuccess = (BOOL) SendMessage(hWnd, HLM_SETTABSTOPS, (WPARAM) iNumber,  
(LPARAM) lpTabs);
```

#### C++

```
Window Style: HLS_USETABS  
bSuccess = [CHListObj.]SetTabstops(iNumber, lpTabs);
```

#### VBX

Not Used

### Arguments/Parameters

<code>int iNumber</code>	The number of tab stops to set
<code>int far *lpTabs</code>	An array of integer TabStops in dialog units

### Return values

<code>BOOL bSuccess</code>	TRUE if TabStops were set correctly
----------------------------	-------------------------------------

### Remarks

If `iNumber` is zero and `lParam` is NULL, the default tab stop is eight dialog units.  
If `iNumber` is 1, the TabStops are spaced evenly based on the first value pointed to by the `lParam`.  
To set and display tabs, the `HLS_USETABS` style or `UseTabs` property must be set at design time.

## Text Attribute

A character string representing the formatted data for each item

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HLM_GETTEXT, iIndex, lpBuf);
```

#### C++

```
lBytesCopied = [CHListObj.]GetText(lpBuf, iIndex);
```

#### VBX

```
[form.][control.]Text
```

### Arguments/Parameters

int iIndex

The index of an item in the List

LPSTR lpBuf

A buffer for the Text

### Return values

lBytesCopied

The actual number of bytes copied to lpBuf

### See Also

[TextColor](#), [TextLen](#)



## TextColor Attribute

The color used when painting the Text

### Usage

#### C

```
crTextColor = (COLORREF)SendMessage(hWnd, HLM_GETTEXTCOLOR, bNegative, 0L);  
SendMessage(hWnd, HLM_SETTEXTCOLOR, bNegative, crNewColor);
```

#### C++

```
crTextColor = [CHListObj.]GetTextColor([bNegative = FALSE]);  
[CHListObj.]SetTextColor(crNewColor [, bNegative = FALSE]);
```

#### VBX

```
[form.][control.]TextColor[= color]  
[form.][control.]TextColor_Neg[= color]
```

### Arguments/Parameters

BOOL bNegative	If TRUE, the TextColor for negative numbers is gotten or set.
COLORREF crNewColor	The new TextColor

### Return values

COLORREF crTextColor	The current TextColor
----------------------	-----------------------

### Remarks

The TextColor attribute can be set at design time or as part of the [WindowText](#).

### See Also

[Text](#), [Format](#)

## TextLen Attribute

The length of the Text in characters

### Usage

#### C

```
iTextLen = SendMessage(hWnd, HLM_GETTEXTLEN, (WPARAM)iIndex, 0L);
```

#### C++

```
iTextLen = [CHListObj.]GetTextLen(iIndex);
```

#### VBX

```
Len([control.]Text)
```

### Arguments/Parameters

int iIndex

The index of an item in the List

### Return values

int iTextLen

The TextLen for item iIndex

### See Also

[Text](#)

## TopIndex Attribute

The index of the item displayed at the top of the List

### Usage

#### C

```
iTop = (int)SendMessage(hWnd, HLM_GETTOPINDEX, 0, 0L);  
iTop = (int)SendMessage(hWnd, HLM_SETTOPINDEX, (WPARAM)iIndex, 0L);
```

#### C++

```
iTop = [CHListObj.]GetTopIndex();  
iTop = [CHListObj.]SetTopIndex(iIndex);
```

#### VBX

```
[form.][control.]TopIndex[= index]
```

### Arguments/Parameters

<code>int iIndex</code>	The index of the new top item
-------------------------	-------------------------------

### Return values

<code>int iTop</code>	The index of the top item
-----------------------	---------------------------

## HList Methods

<u>Add</u>	<u>Insert</u>	<u>SelectItem</u>
<u>Delete</u>	<u>Reset</u>	<u>SelectString</u>
<u>FindCode</u>	<u>Retrieve</u>	
<u>FindData</u>	<u>SelectCode</u>	
<u>FindString</u>	<u>SelectData</u>	

## ListBox and ComboBox Methods

<b>ListBox</b>	<b>ComboBox</b>
<u>Add</u>	<u>Add</u>
<u>Delete</u>	<u>Delete</u>
<u>Insert</u>	<u>Insert</u>
<u>Reset</u>	<u>Reset</u>
<u>Retrieve</u>	<u>Retrieve</u>
<u>SelectItem</u>	<u>Selection</u>

## Add Method

Adds an item or items to a list. If the list is unsorted, addition occurs at the end of the list.

### Usage

#### C

Add a single item

```
iNewIndex = HLAddItem(hWnd, lpData);
```

Add a single item with code

```
iNewIndex = HLAddItemEx(hWnd, lpData, lpCode);
```

Add multiple items

```
iNumber = HLAddItems(hWnd, iCount, lpData);
```

Add multiple items with codes

```
iNumber = HLAddItemsEx(hWnd, iCount, lpData, lpCode);
```

#### C++

Add a single item

```
iNewIndex = [CHListObj.]AddItem(lpData);
```

Add a single item with code

```
iNewIndex = [CHListObj.]AddItemEx(lpData, lpCode);
```

Add multiple items

```
iNumber = [CHListObj.]AddItems(iCount, lpData);
```

Add multiple items with codes

```
iNumber = [CHListObj.]AddItemsEx(iCount, lpData, lpCode);
```

## VBX

Add a single item

```
[form.][control.]AddItem strData
```

Add a single item with code

```
iNewIndex = VLAddItemEx(control.hWnd, strData, strCode)
```

### Arguments/Parameters

int iCount

The number of items to add

void FAR \*lpData

A pointer to the Data item (single) or array (multiple)

strData

A string representing the Data to add

void FAR \*lpCode

A pointer to the Code item (single) or array (multiple)

strCode

A string representing the Code to add

### Return values

int iNewIndex

The index at which an item was added, or an error code

int iNumber

The number of items successfully added, or an error code

### See Also

[Insert](#)

## Delete Method

Deletes an item or items from the List

### Usage

#### C

Delete a single item

```
bSuccess = (BOOL)SendMessage(hWnd, HLM_DELETEITEM, (WPARAM)iIndex, 0L);
```

Delete multiple items

```
iNumber = HLDeleteItems(hWnd, wSearchCat, iCount, lpSearchInfo);
```

#### C++

Delete a single item

```
bSuccess = [CHListObj.]DeleteItem(iIndex);
```

Delete multiple items

```
iNumber = [CHListObj.]DeleteItems(wSearchCat, iCount, lpSearchInfo);
```

#### VBX

```
[form.][control.]RemoveItem iIndex
```

### Arguments/Parameters

<code>int iIndex</code>	The index of an item in the List
<code>WORD wSearchCat</code>	A <u>search category</u>
<code>int iCount</code>	The number of items to find and delete
<code>void far *lpSearchInfo</code>	A pointer to an array of Data items, Codes, or Indices depending on the value of wSearchCat

### Return values

<code>BOOL bSuccess</code>	TRUE if the item was deleted successfully
<code>int iNumber</code>	The number of items successfully deleted

### See Also

Add, Insert

## FindCode Method

Gets the index of an item given its Code

### Usage

#### C

```
iIndex = (int)SendMessage(hWnd, HLM_FINDCODE, (WPARAM)iStart,  
(LPARAM)lpCode);
```

#### C++

```
iIndex = [CHListObj.]FindCode(iStart, lpCode);
```

#### VBX

```
VLFindCode(control(hWnd), iStart, strCode)
```

### Arguments/Parameters

int iStart

The index at which to begin searching

void FAR \*lpCode

A pointer to the Code to search for

strCode

A string representing the Code to search for

### Return values

int iIndex

The index of the item or HLERR\_NOTFOUND if no match was found

### See Also

[FindData](#), [FindString](#)

## FindData Method

Gets the index of an item given its Data

### Usage

#### C

```
iIndex = (int)SendMessage(hWnd, HLM_FINDDATA, (WPARAM)iStart,  
(LPARAM)lpData);
```

#### C++

```
iIndex = [CHListObj.]FindData(iStart, lpData);
```

#### VBX

```
VLFindData (control.hWnd, iStart, strData)
```

### Arguments/Parameters

int iStart

The index at which to begin searching

void FAR \*lpData

A pointer to the Data to search for

strData

A string representing the Data to search for

### Return values

int iIndex

The index of the item or HLERR\_NOTFOUND if no match was found

### See Also

[FindCode](#), [FindString](#)



## FindString Method

Gets the index of an item given some or all of its Text

### Usage

#### C

```
iIndex = (int)SendMessage(hWnd, HLM_FINDSTRING, (WPARAM)iStart,  
(LPARAM)lpText);
```

#### C++

```
iIndex = [CHListObj.]FindString(iStart, lpText);
```

#### VBX

```
VLFindString (control.hWnd, iStart, strText)
```

### Arguments/Parameters

<code>int iStart</code>	The index at which to begin searching
<code>void FAR *lpText</code>	A pointer to the <u>Text</u> to search for
<code>strText</code>	A string representing the <u>Text</u> to search for

### Return values

<code>int iIndex</code>	The index of the item or HLERR_NOTFOUND if no match was found
-------------------------	---

### Remarks

The FindString attribute will attempt to find the closest match when passed a partial string.

### See Also

[FindCode](#), [FindData](#)

## Insert Method

Inserts an item or items into an unsorted List

### Usage

#### C

Insert a single item

```
iNewIndex = HLInsertItem(hWnd, iPos, lpData);
```

Insert a single item with code

```
iNewIndex = HLInsertItemEx(hWnd, iPos, lpData, lpCode);
```

Insert multiple items

```
iNumber = HLInsertItems(hWnd, iPos, iCount, lpData);
```

Insert multiple items with codes

```
iNumber = HLInsertItemsEx(hWnd, iPos, iCount, lpData, lpCode);
```

#### C++

Insert a single item

```
iNewIndex = [CHListObj.]InsertItem(iPos, lpData);
```

Insert a single item with code

```
iNewIndex = [CHListObj.]InsertItemEx(iPos, lpData, lpCode);
```

Insert multiple items

```
iNumber = [CHListObj.]InsertItems(iPos, iCount, lpData);
```

Insert multiple items with codes

```
iNumber = [CHListObj.]InsertItemsEx(iPos, iCount, lpData, lpCode);
```

#### VBX

Insert a single item

```
[form.][control.]AddItem strData, iPos
```

Insert a single item with code

```
iNewIndex = VLInsertItemEx(control.hWnd, iPos, strData, strCode)
```

### Arguments/Parameters

<code>int iPos</code>	The index at which to insert
<code>int iCount</code>	The number of items to insert
<code>void FAR *lpData</code>	A pointer to the Data item (single) or array (multiple)
<code>strData</code>	A string representing the Data to insert
<code>void FAR *lpCode</code>	A pointer to the Code item (single) or array (multiple)
<code>strCode</code>	A string representing the Code to insert

### Return values

<code>int iNewIndex</code>	The index at which an item was inserted, or an <u>error code</u>
<code>int iNumber</code>	The number of items successfully inserted, or an

error code

**See Also**  
Add

## Reset Method

Removes all items from the List

### Usage

#### C

```
SendMessage(hWnd, HLM_RESETCONTENT, 0, 0L);  
bSuccess = HLEmptyList(hWnd);
```

#### C++

```
bSuccess = [CHListObj.]EmptyList();
```

#### VBX

```
[form.][control.]Clear
```

### Return values

BOOL bSuccess

TRUE if the List was emptied successfully

## Retrieve Method

Gets the Data items, Codes, or indices based on a search criterion

### Usage

#### C

```
iNumber = HLGetItems(hWnd, iCount, wReturnCat, lpReturnInfo, wSearchCat, lpSearchInfo)
```

#### C++

```
iNumber = [CHListObj.]GetItems(iCount, wReturnCat, lpReturnInfo, wSearchCat, lpSearchInfo)
```

#### VBX

Not Used

### Arguments/Parameters

```
int iCount  
WORD wReturnCat  
void FAR *lpReturnInfo
```

```
WORD wSearchCat  
void FAR *lpSearchInfo
```

The maximum number of items to find and return

A return category

A pointer to an array of Data items, Codes or Indices, depending on the wReturnCategory to receive the returned data.

A search category

A pointer to an array of Data items, Codes or Indices, depending on the wSearchCategory.

### Return values

```
int iNumber
```

The actual number of items returned

### See Also

[Selection](#), [SelectionMode](#)

## SelectCode Method

Selects an item based on its Code

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HLM_SELECTCODE, wAction, (LPARAM)lpCode);  
HLSelctCode(wAction, lpCode);
```

#### C++

```
bSuccess = [CHListObj.]SelectCode(lpCode, wAction);
```

#### VBX

```
[form.][control.]SelectedCode[ = strCode]
```

### Arguments/Parameters

WORD wAction	An <u>action code</u>
void FAR *lpCode	A pointer to the Code to match
strCode	A string representing the Code to match

### Return Value

BOOL bSuccess	TRUE if selection is successful
---------------	---------------------------------

### See Also

SelectData, SelectionState, SelectItem, SelectString

## SelectData Method

Selects an item based on its Data

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HLM_SELECTDATA, wAction, (LPARAM)lpData);  
HLSelctData(wAction, lpData);
```

#### C++

```
bSuccess = [CHListObj.]SelectData(lpData, wAction);
```

#### VBX

```
[form.][control.]SelectedData[ = strData]
```

### Arguments/Parameters

WORD wAction	An <u>action code</u>
void FAR *lpData	A pointer to the Data to match
strData	A string representing the Data to match

### Return Value

BOOL bSuccess	TRUE if selection is successful
---------------	---------------------------------

### See Also

SelectCode, SelectionState, SelectItem, SelectString

## SelectItem Method

Selects an item based on its index

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HLM_SELECTITEM, wAction, (LPARAM)iIndex);  
bSuccess = HLSelectItem(hWnd, wAction, iIndex);  
iNumber = SendMessage(hWnd, HLM_SELECTRANGE, wAction, MAKELONG(iStart,  
iEnd));  
iNumber = HLSelectItems(hWnd, wAction, wSearchCat, iCount,  
lpSearchInfo);
```

#### C++

```
bSuccess = [CHListObj.] SelectItem(iIndex, wAction);  
iNumber = [CHListObj.] SelectItems(wAction, wSearchCat, iCount,  
lpSearchInfo);
```

#### VBX

```
bSuccess = VLSelectItem(control.hWnd, iIndex, iAction);
```

### Arguments/Parameters

WORD wAction	An <u>action code</u>
int iIndex	The index of the item to select
WORD wSearchCat	A <u>search category</u>
iCount	The maximum number of items to select
void FAR *lpSearchInfo	A pointer to an array of Data items, Codes or Indices, depending on the wSearchCategory.

### Return Value

BOOL bSuccess	TRUE if selection is successful
iNumber	The actual number of items selected

### See Also

[SelectCode](#), [SelectData](#), [SelectionState](#), [SelectString](#)



## SelectString Method

Selects an item based on some or all of its Text

### Usage

#### C

```
bSuccess = SendMessage(hWnd, HLM_SELECTSTRING, (WPARAM)iStart,  
(LPARAM)lpText);
```

#### C++

```
bSuccess = [CHListObj.]SelectString(iStart, lpText);
```

#### VBX

```
bSuccess = VLSelectString(control(hWnd), iStart, lpText);
```

### Arguments/Parameters

int iStart	The index at which to start searching
LPSTR lpText	Complete or partial Text of the item to select

### Return Value

BOOL bSuccess	TRUE if selection is successful
---------------	---------------------------------

### See Also

[SelectCode](#), [SelectData](#), [SelectionMode](#), [SelectItem](#)

## HList Events

DoubleClick

SelectChange

SpaceError

KillFocus

SetFocus

### DoubleClick Event

The user has double-clicked on an item.

#### Usage

##### C/C++

Notification code: **HLN\_DBLCLICK**

##### VBX

**Sub** *ctlname\_DblClick*

#### Return Value

Not Used

## KillFocus Event

The control has lost input focus.

### Usage

#### C/C++

Notification code: **HLN\_KILLFOCUS**

#### VBX

**Sub** *ctlname\_KillFocus*

### Return Value

Not Used

## SelectChange Event

The Selection has changed.

### Usage

#### C/C++

Notification code: **HLN\_SELCHANGE**

#### VBX

**Sub** *ctlname\_SelChange*

### Return Value

Not Used

## SetFocus Event

The control has gained input focus.

### Usage

#### C/C++

Notification code: **HLN\_SETFOCUS**

#### VBX

**Sub** *ctlname\_SetFocus*

### Return Value

Not Used

## SpaceError Event

The control is unable to perform an operation because of memory constraints.

### Usage

#### C/C++

Notification code: **HLN\_ERRSPACE**

#### VBX

**Sub** *ctlname*\_ErrSpace

### Return Value

Not Used

## HList Error Codes

<b><u>Code</u></b>	<b>Description</b>
HLERR_GENERAL	A general error occurred.
HLERR_SPACE	The control was unable to allocate memory.
HLERR_EMPTY	There is no selection.
HLERR_BADVAL	A value was not in the expected group or range.
HLERR_BADPTR	A pointer was found to be NULL unexpectedly.
HLERR_NOCODES	The List does not contain Codes.
HLERR_NOTFOUND	The item was not found.

## HList State Flags

<b><u>Flag</u></b>	<b>Meaning when set</b>
HLF_HASCODES	The control is maintaining a list of codes as well as data for each item
HLF_CHANGED	The selection has changed since it was last set

## Selection Action Codes

<b><u>Flag</u></b>	<b>Action</b>
HL_SELECT	Selects the item
HL_DESELECT	Unselects the item
HL_TOGGLE	Toggles selection state

## HList Search Categories

<b><u>Code</u></b>	<b>Description</b>
HL_DATA	The method searches for Data in the List matching those sent to the method.
HL_CODE	The method searches for Codes in the List matching those sent to the method.
HL_INDEX	The method searches for the List items matching the integer indices sent to the method.
HL_ALL	All items are matched.
HL_SELECTED	All selected items are matched.
HL_UNSELECTED	All unselected items are matched.

## HList Return Categories

<b>Code</b>	<b>Description</b>
HL_DATA	The method returns the items' Data.
HL_CODE	The method returns the items' Codes.
HL_INDEX	The method returns the items' Indices.

### ▣ HList Window Text

This sample window text string is expanded below to show the meaning of each component:

`%ms%ni###-####`

<code>%</code>	Required placeholder.
<code>ms</code>	The <u>DataClass</u> and <u>DataType</u> indicators -- in this case, HC_MASK and HT_STRING.
<code>%</code>	Optional placeholder. Use if the list contains Codes.
<code>ni</code>	The <u>CodeClass</u> and <u>CodeType</u> indicators -- in this case, HC_NUMBER and HT_INTEGER. Optional. Use if the list contains Codes.
<code>###-####</code>	The <u>Format string</u> -- in this case, a seven-digit telephone number.



## HStat, the WinWidgets Static Control

### ■ Attributes

The HStat control is a simple, capable tool for presenting text, bitmaps, icons, and solid and transparent frames. Thoughtful use of HStat controls improves the usability and appearance of the user interface by distinguishing screen components and otherwise decorating bland forms.

### **Additional Topics**

- Using custom resources
- Displaying 256-color bitmaps

## HStat Attributes

<u>Alignment</u>	<u>Foreground</u>	<u>Text</u>
<u>Background</u>	<u>Palette</u>	<u>Transparency</u>
<u>Border</u>	<u>Picture</u>	<u>Type</u>

## Alignment Attribute

Determines the alignment of the Text or Picture on the face of the control. Alignment may be left, right or center (the default).

### Usage

#### C/C++

Window Styles:

HSS\_LEFT

HSS\_RIGHT

#### VBX

[*form.*][*control.*]**Alignment**[= *Left/Right/Center*]

## Background Attribute

The color used to paint the background of an HStat control

### Usage

#### C

```
crBkgnd = (COLORREF)SendMessage(hWnd, HSM_GETBKGNDCOLOR, 0, 0L);  
SendMessage(hWnd, HSM_SETBKGNDCOLOR, 0, crNewColor);
```

#### C++

```
crBkgnd = [CHStatObj.]GetBkgndColor(void);  
[CHStatObj.]SetBkgndColor(crNewColor);
```

#### VBX

```
[form.][control.]BackColor[ = color ]
```

*See Visual Basic Language Reference, "BackColor, ForeColor Properties"*

### Arguments/Parameters

COLORREF crNewColor	New background color
---------------------	----------------------

### Return values

COLORREF crBkgnd	Current background color
------------------	--------------------------

### Remarks

HStat also supports WM\_CTLCOLOR processing; see the Windows SDK documentation

## BorderStyle Attribute

HStat supports four different border styles: none, standard, indented, extruded and bump.

### Usage

#### C/C++

Window Styles:

HSS\_BORDER3D

HSS\_BUMP

HSS\_INDENT

HSS\_EXTRUDE

#### VBX

[*form.*][*control.*]**BorderStyle**[= *None/Standard/Indent/Extrude/Bump*]

### Remarks

The BorderStyle attribute is read-only at run time.

## Foreground Attribute

A color value used to indicate transparency in bitmaps, or the text color

### Usage

#### C

```
crFrgnd = SendMessage(hWnd, HSM_GETFRGNDCOLOR, 0, 0L);  
SendMessage(hWnd, HSM_SETFRGNDCOLOR, 0, (LPARAM) crNewFrgnd);
```

#### C++

```
crFrgnd = [CHStatObj.]GetFrgndColor();  
[CHStatObj.]SetFrgndColor(crNewFrgnd);
```

#### VBX

```
[form.][control.]MaskColor[= color]  
[form.][control.]TextColor[= color]
```

### Arguments/Parameters

<code>COLORREF crNewFrgnd</code>	The new foreground color
----------------------------------	--------------------------

### Return values

<code>COLORREF crFrgnd</code>	The current foreground color
-------------------------------	------------------------------

### Remarks

For HStat controls displaying bitmaps, the Foreground attribute is used to mask portions of the bitmap and is set by default to light green, RGB (0, 255, 0). Masking causes a background color to show through portions of the bitmap, allowing non-rectangular bitmap images in the same manner as Windows icons.

## Palette Attribute

A handle to a 256-color palette for a DIB-type [Picture](#)

### Usage

#### C

```
hpPalette = SendMessage(hWnd, HSM_GETPALETTE, 0, 0L);
```

#### C++

##### OWL

```
hpPalette = [CHStatObj.]GetPalette();
```

##### MFC

```
pPalette = [CHStatObj.]GetPalette();
```

#### VBX

*VBX static controls are "palette aware"; they automatically realize their own palettes.*

### Return values

HPALETTE hpPalette

A handle to a logical palette for the control

CPalette \*pPalette

A pointer to a CPalette object containing the handle to control's logical palette

### Remarks

Typically, an application that displays 256-color images will select and realize the control's palette in response to WM\_PALETTECHANGED and WM\_QUERYNEWPALETTE messages. For additional information see the example below and Microsoft's Palette Self-Study Module, available through the Microsoft Developer Network.

### Examples



## Picture Attribute

The bitmap or icon displayed by the control

### Usage

#### C

```
hPic = (HANDLE)SendMessage(hWnd, HSM_GETPIC, 0, 0L);  
hOldPic = (HANDLE)SendMessage(hWnd, HSM_SETPIC, 0,  
(LPARAM)MAKELONG(hNewPic, wType));
```

#### C++

```
hPic = [CHStatObj.]GetPic();  
hOldPic = [CHStatObj.]SetPic(hNewPic, wType);
```

#### VBX

```
[control.]Picture[= picture]
```

### Arguments/Parameters

HBITMAP/HICON hNewPic  
WORD wType

A handle to the new picture  
Indicates the type of picture. Can be one of the three [PictureType](#) values

### Return values

HBITMAP/HICON hPic  
HBITMAP/HICON hOldPic

The handle of the requested picture  
The handle of the previously set picture

### Remarks

C and C++ applications are responsible for destroying any pictures they add at run time.

At design time, the Picture attribute can be set from within the Dialog Editor, Resource Workshop, or Visual Basic, or picture resources can be specified in the [WindowText](#) directly.

### See Also

[Palette](#)

## Text Attribute

A character string displayed by a static text control. Line breaks can be inserted in the text using the ^ (carat) character. HStat will replace all carats with new line characters (\n).

### Usage

#### C

```
lBytesCopied = SendMessage(hWnd, HSM_GETTEXT, (WPARAM) iMaxBytes,  
(LPARAM) lpBuf);
```

```
SendMessage(hWnd, HSM_SETTEXT, 0, (LPARAM) lpBuf);
```

#### C++

```
lBytesCopied = [CHStatObj.]GetText(lpBuf [, iMaxBytes ==-1]);
```

```
[CHStatObj.]SetText(lpBuf);
```

#### VBX

```
[form.][control.]Text[= stringexpression]
```

### Arguments/Parameters

LPSTR lpBuf

A buffer for the control Text

int iMaxBytes

The maximum number of bytes to copy to lpBuf

### Return values

LONG lBytesCopied

The number of bytes actually copied to lpBuf



## TextIndent Attribute

When set, the static control's Text is displayed with a white highlight to the lower left of each character, giving the text an indented appearance

### Usage

#### C/C++

Window Style: **HSS\_TEXTINDENT**

#### VBX

[*form.*][*control.*]**TextIndent**

### Remarks

This property is read-only at run time.

## Transparency Attribute

The static control does not erase its background, allowing whatever is behind the Text or Picture to show through.

### Usage

#### C/C++

Window Style: HSS\_TRANSPARENT

#### VBX

Not Used

## Type Attribute

Determines the type of static control (i.e. text, groupbox, frame, picture, etc.)

### Usage

#### C/C++

Window Styles:

HSS\_TEXT

HSS\_GROUP

HSS\_FRAME

HSS\_PIC

HSS\_HLINE

HSS\_VLINE

#### VBX

Not Used. The various static types are implemented as separate controls in Visual Basic.

### Remarks

The Type attribute is read-only at run time.

## ❑ HStat Window Text

This sample window text string is expanded below to show the meaning of each component:

```
[gray];[white];winlogo
```

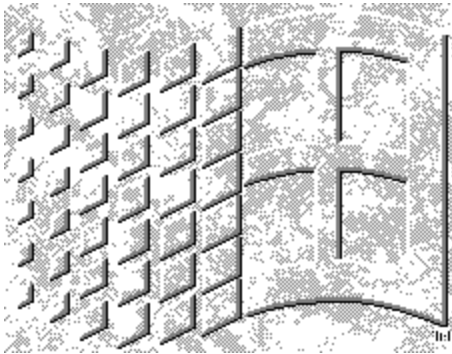
**[gray]** The Foreground color, which is used to paint the text or, in this case, to specify transparent portions of a bitmap.

**;** A required separator.

**[white]** The Background color, which is used to paint the background of the control, including the transparent portions of the bitmap resulting from the Foreground color.

**winlogo** The Picture resource name.

This is the result:



## HTool, The WinWidgets Toolbar

- ▣ Attributes
- ▣ Methods

HTool provides a quick and easy way to create the ribbons, status bars and floating tool palettes that are so popular in today's Windows applications. The toolbar can be designed in a dialog editor like any dialog box, then instantiated using the Create method.

An HTool toolbar can be attached to any side of its parent window or it can float above as a separate window. If the toolbar is attached, the Update method automatically resizes the toolbar to fit its parent. Update is usually called whenever the parent receives a WM\_SIZE message.

The toolbar relays all of the notification messages and WM\_CTLCOLOR messages that it receives from its child controls to its parent window, or to the NotifyWindow if one is specified. In this way, the toolbar is largely transparent to the programmer; its child controls behave like children of the window it notifies.

## HTool Attributes

<u>Alignment</u>	<u>NoStretch</u>	<u>Thickness</u>
<u>Background</u>	<u>NotifyWindow</u>	<u>Type</u>
<u>Caption</u>	<u>Quiet</u>	

## Alignment Attribute

Determines the side of the parent window to which the toolbar is attached. Alignment does not apply to floating palette toolbars.

### Usage

#### C/C++

Window Styles:

HTS\_BOTTOM

HTS\_LEFT

HTS\_RIGHT

HTS\_TOP

#### VBX

[*form.*] [*control.*] **Alignment**

### Remarks

The Alignment attribute is read-only at run time.

## Background Attribute

The color or pattern used to paint the background of floating palette toolbars; attached toolbars are painted with the Windows button-face color.

### Usage

#### C

```
hbrBkgnd = (HBRUSH)SendMessage(hWnd, HTM_GETBRUSH, 0, 0L);  
hbrOldBkgnd = (HBRUSH)SendMessage(hWnd, HBM_SETBRUSH,  
(WPARAM)hNewBrush, 0L);
```

#### C++

##### OWL

```
hbrBkgnd = [THButtObj.]GetBrush(void);  
hbrOldBkgnd = [THButtObj.]SetBrush(hbrNewBrush);
```

##### MFC

```
pBkgnd = [CHButtObj.]GetBrush(void);  
pOldBkgnd = [CHButtObj.]SetBrush(pNewBrush);
```

#### VBX

[form.][control.]**BackColor**[ = color ]

See *Visual Basic Language Reference*, "BackColor, ForeColor Properties"

### Arguments/Parameters

HBRUSH hbrNewBrush	Handle of a new brush to be set as the background brush
CBrush pNewBrush	Pointer to a CBrush object containing the new background brush handle

### Return values

HBRUSH hbrBkgnd	Handle to the current background brush
HBRUSH hbrOldBkgnd	Handle to the previous background brush
CBrush *pBkgnd	Pointer to a CBrush object containing the current background brush handle
CBrush *pOldBkgnd	Pointer to a CBrush object containing the previous background brush handle

### Remarks

C and C++ applications are responsible for destroying any brushes they create. HTool relays all WM\_CTLCOLOR messages received from its child controls to its parent for processing; see the Windows SDK documentation for details

## Caption Attribute

A character string displayed in the title bar of floating palette toolbars

### Usage

#### C

```
SendMessage(hWnd, HTM_GETCAPTION, wCount, (LPARAM)lpBuf);  
SendMessage(hWnd, HTM_SETCAPTION, 0, (LPARAM)lpTitle);
```

#### C++

```
[CHGridObj.]GetCaption(wCount, lpBuf);  
[CHGridObj.]SetCaption(lpTitle);
```

#### VBX

```
[form.][control.]Caption[ = stringexpression]  
See Visual Basic Language Reference, "Caption Property"
```

### Arguments/Parameters

WORD wCount	The number of characters to copy to or from lpBuf
LPSTR lpBuf	A buffer to hold the Title
LPSTR lpTitle	The new Title string



## NoStretch Attribute

When set, an attached toolbar is not be elongated to fill the entire side of its parent window

### Usage

#### C/C++

Window Styles:

HTS\_NOSTRETCH

#### VBX

[*form.*] [*control.*] **NoStretch**

### Remarks

The NoStretch attribute is read-only at run time.

## NotifyWindow Attribute

The handle of the window to which HTool relays child control notifications. By default, the NotifyWindow is the parent of the toolbar.

### Usage

#### C

```
hwndNotify = (HWND)SendMessage(hWnd, HTM_GETNOTIFY, 0, 0L);  
SendMessage(hWnd, HTM_SETNOTIFY, (WPARAM)hwndNew, 0L);
```

#### C++

```
hwndNotify = [CHGridObj.]GetNotify();  
[CHGridObj.]SetNotify(hwndNew);
```

#### VBX

```
[form.][control.]Notify[ = hwndNew]
```

### Arguments/Parameters

HWND hwndNew	Handle of a new NotifyWindow
--------------	------------------------------

### Return values

HWND hwndNotify	Handle of the current NotifyWindow
-----------------	------------------------------------

## Quiet Attribute

When the control is in Quiet mode, it does not relay notification messages to its parent or the [NotifyWindow](#).

### Usage

#### C

```
SendMessage(hWnd, HTM_BEQUIET, bValue, 0L);  
bQuiet = (BOOL)SendMessage(hWnd, HTM_ISQUIET, 0, 0L);
```

#### C++

```
[CHToolObj.]BeQuiet(bValue);  
bQuiet = [CHToolObj.]IsQuiet();
```

#### VBX

```
SendMessage(control(hWnd), HTM_BEQUIET, bValue, 0L)  
bQuiet = SendMessage(control(hWnd), HTM_ISQUIET, 0, 0L)  
See VBX Advanced Topics
```

### Arguments/Parameters

BOOL bValue

TRUE turns on Quiet mode, FALSE turns it off

### Return values

BOOL bIsQuiet

TRUE if the control is in Quiet mode

## Thickness Attribute

The width in pixels of an attached toolbar perpendicular to the side to which it is attached

### Usage

#### C

```
iThickness = (int)SendMessage(hWnd, HTM_GETTHICK, 0, 0L);  
SendMessage(hWnd, HTM_SETTHICK, (WPARAM)hWndNew, 0L);
```

#### C++

```
iThickness = [CHGridObj.]GetThick();  
[CHToolObj.]SetThick(iNewThickness);
```

#### VBX

```
[form.][control.]Thickness [ = iNewThickness]
```

### Arguments/Parameters

int iNewThickness	The new toolbar Thickness in pixels
-------------------	-------------------------------------

### Return values

int iThickness	The current toolbar Thickness in pixels
----------------	---

## Type Attribute

Determines the type of toolbar as either attached or floating

### Usage

**C/C++**

Window Styles:

HTS\_FLOAT

**VBX**

[*form.*] [*control.*] **Type**

### Remarks

The Type attribute is read-only at run time.

## HTool Methods

Create

Update

## Create Method

Creates an instance of the HTool control based on dialog template and style

### Usage

#### C

```
hwndToolBar = HToolCreate (hInst, lpTemplate, hwndParent, hwndNotify,  
dwStyle, wID, iX, iY);
```

#### C++

##### OWL

```
new THTool(AParent, lpTemplate, dwStyle, wId, AModule)  
[THToolObj.]Create()  
[THToolObj.]Create(iX, iY)
```

##### MFC

```
bCreated = [CHToolObj.]Create(lpTemplate, pParent, pNotify=NULL,  
dwStyle=HTS_TOP, wID=-1, iX=0, iY=0);
```

#### VBX

Not Used

### Arguments/Parameters

LPCSTR lpTemplate	A character string containing the name of the toolbar's dialog template resource
HINSTANCE hInst	The handle of the instance containing the toolbar's dialog template resource
CWnd* pParent	The toolbar's parent window
CWnd* pNotify	The toolbar's <u>NotifyWindow</u>
DWORD dwStyle	The toolbar's <u>window style</u> , including the <u>Type</u> , <u>Alignment</u> , <u>NoStretch</u> and Visible attributes
UINT wID	The ID of the toolbar
UINT iX	The initial horizontal location of a floating palette
UINT iY	The initial vertical location of a floating palette
PTWindowsObject AParent	The toolbar's parent window
PTModule AModule	The module containing the toolbar's dialog template resource

### Return values

HWND hwndToolBar	The handle of the toolbar window
BOOL bCreated	A non-zero value if the toolbar was successfully instantiated

## Update Method

Resizes or repositions an attached toolbar to fill the side of its parent to which it is aligned

### Usage

#### C

```
HToolUpdate (hwndToolBar);
```

#### C++

```
[CHToolObj.]Update()
```

#### VBX

Not Used

### Arguments/Parameters

HWND hwndToolBar

The handle of the toolbar window

## ❑ **HTool Window Text**

This sample window text string is expanded below to show the meaning of each component:

MyToolbar

MyToolbar

The name of the dialog template resource from which to construct the toolbar



## **Borland's Object Windows Library (OWL)**

OWL is an extensive C++ class library from Borland that encapsulates most of the Windows API. The WinWidgets Professional Edition includes OWL-compatible classes that encapsulate the WinWidgets' functionality.

## **Microsoft's Foundation Classes (MFC)**

MFC is an extensive C++ class library from Microsoft that encapsulates the Windows API, and provides significant extensions. The WinWidgets Professional Edition includes MFC-compatible classes that encapsulate the WinWidgets' functionality.

## **Custom Controls**

Controls are the components used to create a user interface, such as buttons, scroll bars and list boxes. *Custom controls* are extensions to the set of standard controls provided by Windows. Their purpose is enhance the behavior, appearance or capabilities of the standard controls, or to provide new abilities not addressed by the standard controls.

## **The Windows Application Programming Interface (API)**

The messages and functions defined and provided by the Windows environment and accessible to application programmers.

## **Event Procedures and Notification Codes**

The WinWidgets use event procedures and notification codes to tell an application about user actions, such as pressing a button or editing a cell within the Grid. When these events occur, the WinWidgets send notification codes to their parent window through the WM\_COMMAND message. In Visual Basic, OWL and MFC, the notification codes in turn trigger event procedures. Event procedures can be customized by the programmer to provide the desired behavior for the application. See also: [Handling Events](#)

## Messages

Messages are the most basic means for an application to communicate with the WinWidgets. Each control has a message handling procedure that reads messages from the application message queue. Messages are added to the queue using the *SendMessage()* and *PostMessage()* functions from the Windows API. Each message is simply a predefined constant that causes the control to take a particular action. A message may have associated parameters.

## **Instance**

In order to load a resource, the WinWidgets need a handle to an Instance. Typically, an Instance is an executable (.EXE) or a DLL, and the handle to the Instance is provided through WinMain() or LibMain().

## **Data Tables**

Data tables consist of records (or rows), each of which contains data for a number of fields (columns). A sample table, from a hypothetical personnel system, has a record for each employee that contains a data entry for the fields "ID," "Name," "Position," "Date of Birth," "Date of Hire," and "Salary."



## Callback Procedures

A callback procedure is a function that is implemented in a DLL or executable and exported so that it may be called from another DLL or executable. A function is exported using the `__export` keyword or by declaring the function in the **EXPORTS** section of the module definition (.DEF) file.

## **Windows International Settings**


Windows provides the ability for users to customize their work environment to suit regional conventions or personal tastes. The properties that can be customized include date, time and currency formats (such as "mm/dd/yyyy") and date, time and numeric separators.

## **Buffer Procedure Definition**

A BUFFERPROC is declared as follows:


```
    BOOL FAR PASCAL MyBuffer(HWND, WORD, LONG, LPVOID);
```

## **Popup Topics**

The  button pops up a window with additional information or selection options for a topic. It has the same effect as green text with a dashed underline.

**This page intentionally left blank**

## Jump Buttons

The  button switches to a related topic. It has the same effect as green text with a solid underline. To go back to the previous topic press the **Back** button or the **History** button.

## **The DataEngine**

The DataEngine is a set of routines used by the WinWidgets to convert between illegible binary data and text. The methods used in these conversions depend on the data's classification, or the *data class*. Data are grouped into classes based on their meaning in the real world. For example, Dates, Times, Numbers, and Strings are data classes.

## **Helpful Tips**

The following topics contain general information on using the WinWidgets. We hope you find them helpful.



## Technical Support

We at Simple Software work hard to provide prompt and accurate responses to questions and problems concerning the WinWidgets. However, we will not support non-registered users or multiple users at the same site who do not hold individual licenses. It is also not possible for us to provide advice or guidance in areas not directly related to the WinWidgets. For information on Windows programming, we recommend the following resources:

*Microsoft Developer's Network CD*

*Windows Tech Journal*

*Microsoft Systems Journal*

*Programming Windows, Charles Petzold*

*Power Programming Techniques, Peter Norton & Paul Yao*

*Advanced Windows Programming, Martin Heller*

If a problem or question does not require immediate attention, please send a fax or e-mail rather than calling, including readily-compileable code if possible. To date, we have provided excellent technical support and with your respect for these guidelines we will continue to do so. Without it, we will be forced to introduce WinWidgets Phone Help, "...for information about the Grid, please press 22..."

Simple Software can be reached via:

fax: (718) 965-1740

phone: (718) 965-1710 (1p.m. to 6 p.m. EST - **Do not** call our sales line)

CompuServe: 71542,1502

mail: 543 3rd Street  
Brooklyn, NY 11215

Please consult the [Frequently Asked Questions](#) section of this manual before you call for technical support. We will periodically add to this list and post updated manuals to CompuServe in the Windows SDK forum (GO WINSDK) in the Public Utilities section. Look for the file WDGHLP.ZIP.

## Handling Events

The Windows API, MFC, OWL and Visual Basic each provide a different means of processing events. The following code samples for the HEdit Change notification provide an outline for handling events in each environment. For more information, consult the documentation for the appropriate development environment.

### C API

*A WM\_COMMAND message is sent to the parent of the control. The wParam contains the control ID; lParam contains the control window handle in the LOWORD; and notification code in the HIWORD.*

```
case WM_COMMAND:
    if (HIWORD(lParam) = HEN_CHANGE)
    {
        //Process Change Event
    }
```

### MFC

*An ON\_CONTROL statement is placed in the message map specifying a member function to be called for a given notification code from a specific control.*

```
BEGIN_MESSAGE_MAP(className, BaseClassName)
    ...
    ON_CONTROL(HLN_CHANGE, ID_EDIT, OnChange)
END_MESSAGE_MAP()

afx_msg void className::OnChange()
{
    //Process Change Event
}
```

### OWL

*A member function is declared as below for processing all notification codes for a given control. The LP.Hi member of the function's TMessage argument contains the notification code.*

```
virtual void EditNotifyProc(TMessage& Msg) = [ID_FIRST + Control ID];
    ...
void EditNotifyProc(TMessage& Msg)
{
    switch(Msg.LP.Hi)
    {
        ...
        case HEN_CHANGE:
            //Process Change Event
    }
```

```
}
```

### **Visual Basic**

*In Visual Basic, events trigger control procedures that are declared as shown below (though the arguments will vary for different events):*

```
Sub ctlname_Change (Index As Integer)
```

```
    'Process Change Event
```

```
    :
```

```
End Sub
```

## WinWidgets and the Visual Basic Data Control

In Visual Basic 3.0, the WinWidgets are "Data-Aware." This means the WinWidgets can be connected to a data source that will provide them with data and which they can update, all without any coding. The data source can be any database format supported by the Visual Basic Data Control. Currently, these include Microsoft Access, Btrieve, dBASE, FoxPro, Oracle, Paradox, and Microsoft SQL Server.

### The Edit and Button Controls

Each of the WinWidgets employs the Data Control in its own way. The CheckBox, RadioButton and Edit control are designed to connect to a single database field, and to display and allow editing of the current record's data for that field. Simply specify the control's DataSource (a Data Control) and its DataField (a field in the Data Control's Dynaset). No data will appear at design time.

### The ListBox and ComboBox

Unlike the Edit and Button controls, the ListBox and CheckBox do not modify their data source automatically. They are used to display, but not edit, all of the data for a particular field within a table. Starting with the first record in the table, these controls extract the data from the desired field, add it to their lists, and proceed to the next record. To fill a ListBox or ComboBox in this way, specify the control's DataSource (a Data Control) and its DataField (a field in the Data Control's Dynaset).

This process leaves a conspicuous gap in the "Data-Awareness" of the ListBox and ComboBox. Really, these controls have two sources of data: one to populate their lists and one to set the selection. In order to implement the second data source it is necessary to use a dummy edit control to store the selection. To do this, link a hidden edit control to the "selecting field." Then, whenever the Data Control moves to a new record, use the edit control's new data to select the appropriate list item. Also, whenever the selection in the list changes, update the edit control, which will in turn update the database.

It is also possible to use the Codes feature of the ListBox and ComboBox with the Data Control. Simply specify the name of the field from the data source in the CodeField property of the control. This name must be entered manually; unlike the DataField property, editing the CodeField property will not present a list of appropriate field names.

### The Grid

The Grid is used to display and edit complete database tables, or subsets of tables. The easiest way to fill a Grid control is to specify its DataSource (a Data Control), which should be connected to a database table or the result of an SQL query. The Fields property of the Grid need not be set. At run-time, the Grid determines the number and type of fields in its DataSource and creates appropriate columns. Whenever the user edits data, the Grid updates its DataSource. When connected to the Data Control, the Grid implements a record buffer automatically so that only a portion of the entire table is displayed at once.

Alternatively, the Fields property can be initialized at design time if the default behavior is not suitable. If the Fields are initialized, they must be designed to match the DataSource both in number and data type. Typical customizations include the Field's control type, format string, hidden and browse properties.

**Tip:** To control the selection of fields from a table, enter an SQL Select statement in the Data Control's RecordSource property, such as "SELECT EmpID, LastName, FirstName FROM Employee."

The Grid also provides an elegant way to join tables, using the Codes feature of the ListBox and ComboBox controls. It is common in relational database design to store objects of a

common type in one table, giving each a unique ID. Then, wherever an object of that type is referenced elsewhere, only the ID needs to be stored. Here is an example of how to exploit this common design in the Grid:

Assume a database of publications has tables for Titles, Authors and Publishers, and each record in the Titles table has the ID of an Author and a Publisher. To present the table of Titles in a Grid, assign DropList controls (a non-editable ComboBox) to the AuthorID and PublisherID columns. Upon initialization, fill the DropList controls with Author and Publisher names from their respective tables and store the ID's for each name as Codes\*. Also, specify in the Fields Property dialog that these fields are SelectByCodes. When the Grid loads a record from the Titles table, it will match the entry in the AuthorID field to a code in that field's DropList, causing the author's name to be displayed in the cell. If the user has editing privileges, selecting a different Authors name from the list will change the ID stored in the record and cause the Titles table to be updated.

\* To initialize list fields in the Grid from a separate query, create a "dummy" ListBox on your form. Connect the dummy control to the DataControl that contains the auxiliary query (i.e. "Select AuthorName, AuthorID from Authors") and specify both the DataField and CodeField. When the dummy control triggers a Filled event, indicating that it has loaded all of the data, use the VCCopyList() function to transfer the list into the Grid's field control. See the Grid form in the VBDemo application for the complete syntax.

## Using Codes in the ListBox and ComboBox

The WinWidgets ComboBox and ListBox controls both allow non-displayed data to be associated with each item in their lists. The non-displayed data are called *codes* throughout this documentation. Like the displayed list items, the codes may be of any data class and data type supported by the DataEngine. The ListBox and ComboBox maintain the code information using the WinWidgets' memory manager, so it is unnecessary for the application to keep its own copy. In both controls, the codes can be used to order the list (see the SortMode attribute), or to select, delete or retrieve information from the list.

In the HGrid control, a field can be assigned a ListBox or ComboBox to present and edit the data of cells in that field. Using the codes feature of the ListBox and ComboBox, records in the Grid can store code information (such as an ID), while the field control presents the user with a more intelligible list (such as a list of names).

For example, a well-designed database might contain two tables: one of Products, each with a ProductName and a unique ProductID, and the other of Orders, each with an OrderDate, a ProductID and an OrderID. To construct a Grid to edit the Orders table, a ComboBox can be assigned to the ProductID field. The ComboBox is filled from the Products table, using the ProductNames as the displayed data and the ProductID's as the hidden codes. The Grid automatically matches the ProductID from each Order record with the appropriate ProductName, which is displayed.

## Hot-Linking the WinWidgets to Data

The WinWidgets' Button, Edit, ListBox, ComboBox and Grid controls can each be connected to a variable in an application that they update automatically in response to user actions. Hot-Linking avoids the need to poll the controls for their current data or states because that information is updated immediately when it changes. This is particularly useful when elements of a form or dialog are interdependent. Hot-Linking is generally not applicable to Visual Basic programs.

The CheckBox can be linked to any data type supported by the [Boolean data class](#).

The RadioButton can be linked to a WORD variable that will be updated with either the ID of the control or its index in the radiobutton group.

The Edit control can be linked to any data type supported by the [DataEngine](#). It is important to note that when the Edit control is Hot-Linked, the linked variable is not updated until the control parses its text -- normally upon losing focus. When used in a dialog with default pushbuttons or menus, the user can initiate actions after editing but without the control losing focus. In such situations, it is wise to set the focus to another control or [update](#) the Edit control before using the linked variable.

The ListBox and ComboBox can be linked to any data type supported by the [DataEngine](#). Both controls support links to the [Data](#) and/or [Code](#) attributes. The links are updated with the Data or Code for the current [HList, Selection](#) and [HComb, Selection](#).

The Grid supports Hot-Linking to a record buffer. The record buffer is updated with the active record (see [Selection](#)).

## Using Custom Resources

Several of the WinWidgets load resources while they are being created. For example, the CheckBox loads two bitmaps for each of its two states, and the Grid loads a grid-definition resource. There are several places the WinWidgets look to load a resource, and the search list is ordered as follows:

- 1) **[NOT YET IMPLEMENTED]** The parent window of the control receives a HM\_GETINSTANCE message from the WinWidgets, which should return a valid instance handle if the resource is to be loaded from an instance other than the executable creating the control. Otherwise, the parent must return NULL. The WinWidgets attempt to load the resource from the returned instance if it is not NULL.
- 2) If the resource was not loaded as a result of Step #1, the WinWidgets attempt to load it from the executable that created the control.
- 3) If not successful, the WinWidgets attempt to load it from WIDGETS.DLL.
- 4) If not successful, the WinWidgets look for a file with the same name as the resource, using a default extension (.BMP, .ICO, etc.) if necessary.
- 5) A default resource or no resource will be used, as appropriate.



## HM\_GETINSTANCE

The HM\_GETINSTANCE message is used to retrieve a handle to an instance from which the WinWidgets will attempt to load a resource.

### C API

An HM\_GETINSTANCE message is sent to the parent of the control. The wParam contains the control ID; lParam contains the control window handle in the LOWORD; and a resource-type code in the HIWORD.

```
case HM_GETINSTANCE:
    switch (wParam) // Control ID
    {
        case ID_MYBUTTON:
            if (HIWORD(lParam) == RT_BITMAP ||
                HIWORD(lParam) == RT_ICON)
            {
                // Return hInstance for Resource DLL
                return hInstResDLL;
            }
            break;
    }
    return NULL;
```

### MFC

### OWL

### Visual Basic

## Advanced Visual Basic

Although Visual Basic does not provide direct support for the entire Windows programming interface, many additional functions can be accessed simply by declaring their syntax appropriately and the DLL in which they are contained. One of the most important Windows functions is `SendMessage()`, which is widely used by C/C++ programmers to communicate with controls and other windows. `SendMessage` is declared in Visual Basic as follows:

```
Declare Function SendMessage Lib "User" (ByVal hWnd As Integer, ByVal wParam As Integer,
                                         ByVal lParam As Integer, lParam As Any) As Long
```

This declaration and many others are included in `WIN30API.TXT`, which is provided with the Visual Basic Professional Edition. Once `SendMessage()` is defined, VB programmers can access much of the WinWidgets programming interface in the same way C/C++ programmers do. For more information about matching argument types in calls to Windows API functions, see the *Calling Procedures in DLL's* chapter of the *Visual Basic Programmer's Guide*.

# Using the WinWidgets in Visual C++

## Definition of terms

**VBX Controls:** Controls that conform to the Visual Basic specification for custom controls at run-time and in the design environment. Microsoft Foundation Classes only support the Visual Basic Version 1.0 standard; Visual Basic, itself, is now in version 3.0. Since VBX is a standard for Visual Basic, VBX controls cannot provide the low-level access to data and callback procedures to which C and C++ programmers are accustomed.

**Old Style Controls:** The controls that conform to the old standard for custom controls. These custom controls do not work in the Visual Basic design environment, but do work well in the old SDK dialog editor and Borland's Resource Workshop. Since this standard is based directly on the Windows API, old style controls provide full access to data and callback procedures.

**Wrapper Classes:** A set of C++ classes compatible with the Microsoft Foundation Class library (MFC) provided by Simple Software for use with WinWidgets. These classes encapsulate the functionality of WinWidgets at the level of the Windows API. These classes function similarly to the MFC classes that encapsulate the standard Windows edit control, button, listbox, and combobox.

**CVBControl:** A C++ class provided by Microsoft as part of MFC that encapsulates *any* VBX control at the level of the VBX standard. In order to use this class with a VBX control, you must know how VBX controls work in general and how the particular control you are using functions within the VBX standard.

## The Problem

Microsoft Visual C++ is accompanied by two capable, though flawed, resource tools -- the AppStudio and ClassWizard. These tools are interesting because they support Visual Basic (VBX) controls and provide a means of linking controls to application procedures and variables. They are flawed because they provide *extremely limited* support for old style custom controls, and further because they only support VBX version 1.0 controls, while Visual Basic is in release 3.0. Also, because the VBX standard was developed for Visual Basic, most VBX controls do not provide the low-level control familiar to C and C++ programmers.

As the developer, you must make a choice when you begin designing your application whether to use VBX controls and, if so, to what extent. If you choose not to use VBX controls at all, **you should not use AppStudio** to design your forms. Instead, use one of the other resource editors that support old style custom controls fully. These include Borland's Resource Workshop (our recommendation), or the old SDK Dialog Editor (though not the QuickC version). The rest of this topic summarizes your options for using WinWidgets with Visual C++.

## Solutions

C++ developers can use WinWidgets under Visual C++ in one of three ways.

**1. Use VBX versions of WinWidgets in AppStudio and use the CVBControl class in the application code to communicate with the controls and to respond to control events. If you choose this option, you should look at the sections of this manual labeled "VBX" in each topic for information about control properties and events. For more general information about using VBX controls in MFC, see the topic in this manual, "Using VBX Controls in MFC: A General Discussion."**

Advantages

VBX controls are fully compatible with AppStudio, Class Wizard, and Visual C++ so they can be used with all three tools.

#### Disadvantages

You must redistribute *both* WIDGETS.DLL and WIDGEVB.VBX with your application.

VBX controls do not allow access to certain low-level features of WinWidgets, such as Data Validation Procedures, Hot Linking of program data with the control, and Record buffering in the grid. If you wish to use any of these features, you cannot use VBX controls.

**2. Use VBX versions of WinWidgets in AppStudio but connect them to the MFC Class Wrappers in the application code. If you choose this option, you should look at the sections of this manual labeled "C++" in each topic for information about the Wrapper member functions and control events. You must attach the Wrapper classes to the controls by using their AttachToVBX() member functions and detach them by using the overloaded Detach() member functions.**

#### Advantages:

You can design with VBX controls in AppStudio and retain low-level access to the controls by using the Wrapper classes.

#### Disadvantages:

You must redistribute *both* WIDGETS.DLL and WIDGEVB.VBX with your application.

Class Wizard does not work with the Wrapper Classes. It cannot add instances of these classes to your code and it cannot set up DDX/DDV routines for data associated with them.

AttachToVBX() is a hack. When you attach a CWnd-derived Wrapper object to a control using this function, you can use all of the Wrapper object's member functions documented in this manual. You *cannot*, however, use this object to call CWnd member functions (other than Detach()). This is because AttachToVBX(), unlike Attach(), does not allow MFC to make an entry in its permanent lookup table. The MFC function CWnd::FromHandlePermanent() will not return a pointer to the Wrapper object.

**3. Use the the old SDK Dialog Editor or Borland's Resource Workshop to design your forms and connect the controls to MFC Class Wrappers in the application code. If you choose this option, you should look at the sections of this manual labeled "C++" in each topic for information about the Wrapper member functions and control events. Attach the Wrapper classes to the controls by using their Attach() member functions and detach them by using Detach(). If you create controls dynamically by calling a Wrapper object's Create() member function, an old style custom control will be created.**

#### Advantages:

Your Wrapper objects can be treated as full-fledged CWnd objects. That is, they will be registered in MFC's permanent object lookup table and can call any CWnd member function without unfortunate results.

Your application will not have the overhead associated with MFC's VBX interface.

You do not have to redistribute WIDGEVB.VBX with your application, just WIDGETS.DLL.

#### Disadvantages:

You cannot use AppStudio/ClassWizard as a design tool for your forms. (Some would argue that this is not really a disadvantage.)

You cannot use DDX/DDV routines. Try Hot-Linking instead.

Which one of the three solutions you choose will depend on your own personal preferences and the needs of your application. The sample projects in the SAMPLES\MFCGUIDE subdirectory demonstrate each of the three methods.

C programmers using VC++ will find the lack of support for custom controls in AppStudio particularly frustrating, because they will be unable to use VBX controls at all. In this case, we recommend using DLGEDIT.EXE, which is distributed with the SDK, or purchasing Borland's Resource Workshop (for about \$50). Otherwise, the descriptions of the Window Text syntax and the hexadecimal values of style flags for each of the WinWidgets make it possible to design in AppStudio, though the controls will not appear or behave as they do in an application.

## Using the MFC Wrapper Classes

When an application's needs outstrip the capabilities of the VBX interface, the WinWidgets allow complete and direct access to the controls' attributes and methods through tailored C++ classes. These classes are called MFC Wrappers because they are fully compatible with the Microsoft Foundation Classes (MFC), and they completely encapsulate the WinWidgets API. The Wrappers are contained in two statically linkable libraries MFCWDGSM.LIB and MFCWDGSL.LIB for medium and large model, respectively.

When writing and compiling code using the MFC class wrappers, it is important to remember the following:

1. The class wrapper libraries (MFCWDGSL and MFCWDGSM) are not a substitute for WIDGETS.DLL. The libraries only contain the wrapper code around the basic Widgets functionality implemented in WIDGETS.DLL. You must call WidgetsInit() when you initialize your application, you must link to WIDGETS.LIB, and you must also redistribute WIDGETS.DLL with your application.
2. The libraries MFCWDGSL.LIB and MFCWDGSM.LIB installed by our set-up utility are release versions and do not contain debugging information. We did this to conserve space on distribution disks. We have provided makefiles so that you can recompile these libraries with debugging information as needed.
3. Class definitions for the Wrappers are contained in MFCWIDG.H, which you must include in your source.

## How To Use the Wrapper Classes

The MFC Wrappers are an efficient alternative to the generic VBX control interface provided by Microsoft (see the MFC documentation of CVBControl for a complete discussion). We recommend that developers who wish to use the Wrappers continue to make use of the VBX implementation in AppStudio when designing dialog boxes, but make limited use of the ClassWizard's "Edit Variables" feature. Within the application code itself, instances of CVBControl are unnecessary; instances of our classes and their derivatives can be declared and connected to the dialog items via the AttachToVbx() member function. When creating controls dynamically, simply declare an instance of the Wrapper and call the Create() member function.

The following is a list of the classes defined in MFCWIDG.H:

<b><u>Class Name</u></b>	<b>Description</b>	<b>WinWidget</b>	<b>Derived From</b>
CHButt	Generic Pushbutton	HButt	CWnd
CHBCheck	CheckBox	HButt	CHButt
CHB3State	3 State Button	HButt	CHButt
CHBDefault	Defpushbutton	HButt	CHButt
CHBRadio	Radio Button	HButt	CHBCheck
CHComb	ComboBox	HComb	CWnd
CHEdit	Edit control	HEdit	CWnd
CHGrid	Grid control child	HGrid	CWnd

CHGridView	Grid control view	HGrid	CView
CHGridBuffer	Grid buffer object	N/A	CObject
CBufferManager	Manages buffer support for all grids	N/A	CObject
CHList	List control	HList	CWnd
CHStat	Static control	HStat	CWnd
CHTool	Toolbar	HTool	CDialog

## CWnd-derived Controls

WinWidgets custom controls can be created directly from the MFC objects using the Create member function or can be attached to an object using the Attach() or AttachToVBX() member functions. Note that we have added additional Attach functions taking a parent window and a control ID for use with dialog templates. Messages that can be sent to the controls are encapsulated in member functions. Notifications from the controls can be processed by including ON\_CONTROL statements in a message map, specifying the control ID and the notification to process. The list and combobox classes, CHList and CHComb are designed to be subclassed by the application. Both classes contain a virtual function,

```
void Initialize(void).
```

This function is called automatically within the Create member function of the class and can be called at other times. It should be overridden to initialize the list elements. In the CPPDEMO project, MYCOMBO.CPP and MYCOMBO.H are an example of a subclassed combobox. LISTFORM.CPP provides an example of using a CHList object without subclassing.

**WARNING:** While AttachToVBX() is required to attach an instance of our Wrapper classes to a VBX control, it will not create an entry in MFC's permanent lookup table. Objects attached using AttachToVBX() should only be used to call the WinWidgets message wrappers. These objects will not function correctly as CWnd-derived objects. Wrapper objects attached to non-VBX custom controls using the Attach() member function, however, will function normally as CWnd-derived objects.

## Grids

Because of the unusual nature of the grid control, we have provided both a CWnd-derived class (CHGrid) for managing child windows and a CView-derived class (CHGridView) for use with MDI, splitter windows, etc. Like CHList and CHComb, CHGrid contains a virtual Initialize() function that can be overridden to initialize the grid data. Note that when using a grid buffer (described below), initialization of the grid's data will be handled by the buffer procedures, although any list data for drop-down list or combobox fields must be set in Initialize().

The CHGridView class contains a pointer to a grid (*m\_pGrid*) through which the CHGrid member functions can be called. The CView-derived class also has virtual functions for processing all HGrid notification messages. These functions can be overridden when CHGridView is subclassed in an application. The CHGridView class cannot be attached to an existing grid control. CHGridView contains a virtual function OnPrepareGridName(), which should return the name of a grid resource. CHGridView will use this resource to create the grid dynamically.

In addition to these two classes, we have provided the CHGridBuffer, which should be

subclassed to provide access to HGrid's buffering capabilities.



## Using VBX Controls in MFC: A General Discussion

VBX controls have "properties," which identify their attributes. In this manual, we use the word "attribute" to refer to the characteristics of any control, not just a VBX control. For a VBX control, however, property and attribute are synonymous. Each type of control has its own unique set of properties. Property values are stored for each control that is created.

Most properties can be set at any time during program execution, although some can only be set at design-time, as a means of initializing the state of the control. Properties of a control can be set in AppStudio when designing forms. Public member functions in the MFC class, `CVBControl`, are used to get and set properties at run-time. Refer to the MFC documentation for more help on `CVBControl`, and the `VBCIRCLE` sample application in the MFC samples directory.

VBX controls also generate "events," which is the control's way of notifying the application about changes in its state. Each type of control has a unique set of events that it is capable of generating. In MFC, an application must register all control events that it wishes to process. Registration is done by calling `AfxRegisterVBEvent()`. Events can be registered through ClassWizard, and member functions can be created to handle them. Refer to the MFC documentation for more help on `AfxRegisterVBEvent()`, and the `VBCIRCLE` sample application in the MFC samples directory.

In order to use VBX controls, your application must call `CWinApp::EnableVBX` in the `InitInstance()` of your application object. Using AppWizard with the "Custom VBX Controls" box checked will add this code. VBX controls can be created dynamically by calling the `Create()` member function for `CVBControl`, or can be added to dialog templates in AppStudio.

## Subclassing the WinWidgets

Any of the WinWidgets can be easily subclassed to provide custom behavior in a single control or throughout an application. To subclass an individual control, use the **SubclassWW()** procedure, which is exported from WIDGETS.DLL. This procedure replaces the window procedure of a specific control with one provided by the application. It returns the address of the control's current window procedure, which should be called for any messages the new procedure does not fully replace. SubclassWW() does not affect any controls other than the one specified. The following code subclasses an edit control in a dialog box:

```
WNDPROC DefHEditProc;
```

```
LRESULT FAR PASCAL MyFilter (HWND hwnd, UINT msg, WPARAM wp, LPARAM lp)
{
    switch (msg)
    {
        case WM_KEYDOWN:
            if (wp == VK_F1)
                WinHelp (...);
            break;
    }
    return DefHEditProc (hwnd, msg, wp, lp)
}
```

```
BOOL FAR PASCAL MyDialog (HWND hDlg, UINT msg, WPARAM wp, LPARAM lp)
{
    static WNDPROC lpfnMyFilter;
    HWND hwndEdit;
    switch (msg)
    {
        case WM_INITDIALOG:
            {
                hwndEdit = GetDlgItem (hDlg, IDC_EDIT1);
                lpfnMyFilter = MakeProcInstance (MyFilter, hInstance);
                DefHEditProc = SubclassWW (hwndEdit, lpfnMyFilter);
                break;
            }

        case WM_DESTROY:
            FreeProcInstance (lpfnMyFilter);
            DefHEditProc = NULL;
            break;
    }
}
```

```
}  
}
```

To alter the behavior of all controls of a particular type (e.g. all buttons) it is more efficient to create a new window class than to subclass each control individually. To create a new class based on one of the WinWidgets, use the following procedure:

- 1) Call **WidgetsInit()** so that all of the WinWidgets are registered.
- 2) Get the class information for the WinWidgets control using **GetClassInfo()** and **GetWidgetsInst()**.
- 3) Store the window procedure for the WinWidgets control, then replace it in the **WNDCLASS** structure, along with the class name.
- 4) Register the new class with **RegisterClass()**;

The difficulty with this method is that none of the resource editors will know about the new class, making it necessary to replace the control class names in dialog resources after they are designed, or to create all controls explicitly using **CreateWindow()**.

## Window Styles

A window style is a long integer value containing up to 32 behavior and appearance flags for the window. The highest 16 bits are standard Windows styles, such as `WS_POPUP` for pop-up windows and `WS_CHILD` for child windows; the meaning of the lower 16 bits depends on the type of control.

The window style is stored along with the window's text and positioning information in a dialog resource. Normally, the style value is expanded to show each bit that is set as a text constant, such as `HLS_MULTICOL`. The window style is also a parameter of the `CreateWindow()` procedure, which can be called directly to create windows that are not a part of a dialog.

The WinWidgets use the window style that is passed with the `WM_NCCREATE` message to initialize various attributes. After initialization, the window style is not referenced, so calls to `SetWindowLong (... , GWL_STYLE, ...)` will have no effect.

## C Quick Reference



**Button Controls**



**The ComboBox**



**The Edit Control**



**The Grid**



**The List Control**



**Static Controls**



**The ToolBar**

Styles

Messages

Notifications

WindowText

Styles

Messages

Methods

Notifications

WindowText

Styles

Messages

Notifications

WindowText

Styles

Messages

Methods

Notifications

WindowText



Styles

Messages

Methods

Notifications

WindowText

Styles

Messages

WindowText

Styles

Messages

Methods

WindowText

## HEdit Messages

<b>Message</b>	<b>wParam</b>	<b>lParam</b>	<b>Return Value</b>
<u>HEM_BEQUIET</u>	bQuiet	0L	<i>Not Used</i>
<u>HEM_GETBKGNDBRUSH</u>	0	0L	hBrush
<u>HEM_GETDATA</u>	iMaxBytes	lpData	lBytesCopied
<u>HEM_GETDATACLASS</u>	0	0L	cDataClass
<u>HEM_GETDATALINK</u>	0	0L	lpLink
<u>HEM_GETDATASIZE</u>	0	0L	iSize
<u>HEM_GETDATATYPE</u>	0	0L	cDataType
<u>HEM_GETFONT</u>	0	0L	hfFont
<u>HEM_GETFORMAT</u>	iMaxBytes	lpstrBuf	lBytesCopied
<u>HEM_GETHILITEBRUSH</u>	0	0L	hbHilite
<u>HEM_GETMAXTEXTLEN</u>	0	0L	iMaxLen
<u>HEM_GETOVERWRITEMODE</u>	0	0L	bOverwrite
<u>HEM_GETPASSWORDCHAR</u>	0	0L	cPwdChar
<u>HEM_GETSCROLLPOS</u>	0	0L	iScrollPos
<u>HEM_GETSEL</u>	0	0L	lSel
<u>HEM_GETSELTEXT</u>	iMaxBytes	lpstrSelText	<i>Not Used</i>
<u>HEM_GETSTATE</u>	0	0L	lState
<u>HEM_GETTEXT</u>	iMaxBytes	lpBuf	lBytesCopied
<u>HEM_GETTEXTCOLOR</u>	bNeg	0L	crTextColor
<u>HEM_GETTEXTLEN</u>	0	0L	lTextLen
<u>HEM_GETVALIDATE</u>	0	0L	lpfnValProc
<u>HEM_HASCHANGED</u>	0	0L	bChanged
<u>HEM_ISQUIET</u>	0	0L	bQuiet
<u>HEM_REPLACESEL</u>	0	lpStr	<i>Not Used</i>
<u>HEM_SETBKGNDBRUSH</u>	hbrNewBrush	0L	hbrOldBkgnd
<u>HEM_SETCHANGED</u>	bVal	0L	<i>Not Used</i>
<u>HEM_SETDATA</u>	0	lpData	lBytesCopied
<u>HEM_SETDATALINK</u>	0	lpBuf	lBytesCopied
<u>HEM_SETFONT</u>	hfFont	bRedraw	hfOldFont
<u>HEM_SETFORMAT</u>	bRedraw	lpstrBuf	lBytesCopied
<u>HEM_SETHILITEBRUSH</u>	hbrNewHilite	0L	hbrOldHilite
<u>HEM_SETMAXTEXTLEN</u>	iLen	0L	<i>Not Used</i>
<u>HEM_SETOVERWRITEMODE</u>	bMode	0L	<i>Not Used</i>
<u>HEM_SETPASSWORDCHAR</u>	cChar	0L	<i>Not Used</i>
<u>HEM_SETSCROLLPOS</u>	iScroll	bRedraw	iScrollPos
<u>HEM_SETSEL</u>	0	lNewSel	<i>Not Used</i>
<u>HEM_SETTEXT</u>	0	lpBuf	bResult
<u>HEM_SETTEXTCOLOR</u>	bNeg	crTextColor	<i>Not Used</i>
<u>HEM_SETVALIDATE</u>	0	lpfnNewValProc	<i>Not Used</i>

<u>HEM_UPDATE</u>	bParseText	bUpdateText	<i>Not Used</i>
<u>HEM_VALIDATE</u>	0	0L	iResult
<u>WM_CLEAR*</u>	0	0L	<i>Not Used</i>
<u>WM_COPY*</u>	0	0L	<i>Not Used</i>
<u>WM_CUT*</u>	0	0L	<i>Not Used</i>
<u>WM_PASTE*</u>	0	0L	<i>Not Used</i>
<u>WM_UNDO*</u>	0	0L	<i>Not Used</i>





\* Standard Windows messages














## HButton Messages

Message	wParam	lParam	Return Value
<u>HBM_GETBKGNDBRUSH</u>	0	0L	hBrush
<u>HBM_GETCOUNT</u>	0	0L	iCount
<u>HBM_GETDATA</u>	0	lpData	lBytesCopied
<u>HBM_GETDATALINK</u>	0	0L	lpLink
<u>HBM_GETDATASIZE</u>	0	0L	iSize
<u>HBM_GETFONT</u>	0	0L	hFont
<u>HBM_GETMASKCOLOR</u>	0	0L	crMaskColor
<u>HBM_GETPALETTE</u>	0	0L	hPalette
<u>HBM_GETPALIGN</u>	0	0L	iAlign
<u>HBM_GETPIC</u>	iIndex	0L	hPic
<u>HBM_GETRALIGN</u>	0	0L	iRelAlign
<u>HBM_GETSOUND</u>	0	0L	hSound
<u>HBM_GETSTATE</u>	0	0L	iState
<u>HBM_GETSTATECOUNT</u>	0	0L	iStateCount
<u>HBM_GETTALIGN</u>	0	0L	iAlign
<u>HBM_GETTEXT</u>	iMaxBytes	lpText	lBytesCopied
<u>HBM_GETTEXTCOLOR</u>	0	0L	crTextColor
<u>HBM_SETBKGNDBRUSH</u>	hNewBrush	0L	hPreviousBrush
<u>HBM_SETDATA</u>	0	lpData	lBytesCopied
<u>HBM_SETDATALINK</u>	0	lpLink	lBytesCopied
<u>HBM_SETFONT</u>	hNewFont	0L	hPreviousFont
<u>HBM_SETMASKCOLOR</u>	0	crMaskColor	<i>Not Used</i>
<u>HBM_SETPALIGN</u>	iAlign	0L	<i>Not Used</i>
<u>HBM_SETPIC</u>	iIndex	(hNewPic, wType)	hPreviousPic
<u>HBM_SETRALIGN</u>	0	iRelAlign	<i>Not Used</i>
<u>HBM_SETSOUND</u>	hNewSound	0L	<i>Not Used</i>
<u>HBM_SETSTATE</u>	iNewState	bRedraw	<i>Not Used</i>
<u>HBM_SETTALIGN</u>	0	iAlign	<i>Not Used</i>
<u>HBM_SETTEXT</u>	0	lpText	<i>Not Used</i>
<u>HBM_SETTEXTCOLOR</u>	0	crColor	<i>Not Used</i>



## HButton Styles






Constant	Value	Description
 HBS_PUSHBUTTON	0x00L	Appears and behaves as a push button.
 HBS_DEFPUSHBUTTON	0x01L	Appears and behaves as default button.
 HBS_CHECKBOX	0x02L	Appears and behaves as a two state check box.
 HBS_RADIOBUTTON	0x03L	Appears and behaves as a radio









	HBS_3STATE	0x04L	button. Appears and behaves as a three state check box.
	HBS_OWNERDRAW	0x05L	Appears and behaves as a push button.
	HBS_GROUPPUSH	0x06L	Appears as a push button; behaves as a radio button.
	HBS_TRANSPARENT	0x0010L	The button does not erase its background, allowing whatever is behind it to show through.
	HBS_LJUST	0x0020L	The button's Text is left justified. (Only significant with multi-line Text)
	HBS_RJUST	0x0040L	The button's Text is right justified. (Only significant with multi-line Text)
	HBS_NOFOCUS	0x0080L	The button does not take the input focus when pressed with the mouse. Also removes the WS_TABSTOP style.
	HBS_DOWNPICS	0x0400L	There are two pictures for each state, one for pressed and one for unpressed.
	HBS_AUTOADVANCE	0x0800L	The button advances one state in its state cycle each time it is pressed.
	HBS_NOBUTTON	0x1000L	The button is not drawn as a pushbutton, leaving the background as the system COLOR_WINDOW.
	HBS_TEXTINDENT	0x2000L	The Text is painted to give the impression of three dimensions.
	HBS_SQUARED	0x4000L	The corners of the button are drawn square rather than rounded.
	HBS_ASYNC	0x8000L	Any Sound associated with the button is played asynchronously.



## HEdit Styles










### Constant

	Value	Description	
	HES_DISPLAYLEFT	0x0000L	Text is left justified in Display mode [Default].
	HES_EDITLEFT	0x0000L	Text is left justified in Edit mode [Default].
	HES_DISPLAYCENTER	0x0001L	Text is centered in Display mode.
	HES_DISPLAYRIGHT	0x0002L	Text is right justified in Display mode.
	HES_UPPERCASE	0x0008L	All text is converted to uppercase.



	HES_LOWERCASE	0x0010L	All text is converted to lowercase.
	HES_PASSWORD	0x0020L	All characters appear as the password character.
	HES_AUTOHSCROLL	0x0080L	Text scrolls horizontally automatically when the caret nears either end of the window.
	HES_NOHIDESEL	0x0100L	The selection remains displayed when the control loses input focus
	HES_BORDER3D	0x0200L	The border is displayed with a 3D style. If the WS_BORDER bit is set, the control is extruded like a button, otherwise it is indented.
	HES_HILITE	0x0400L	The HiliteBrush is used to paint the background when the control receives the input focus. If the HiliteBrush is not set, a white brush is used.
	HES_EDITRIGHT	0x0800L	Text is right justified in Edit mode.
	HES_HUNGRY	0x1000L	The control swallows Enter and Esc keyboard messages and notifies its parent.



## HList Styles










<b>Constant</b>	<b>Value</b>	<b>Description</b>	
	HLS_SORTBYDATA	0x0002L	The list is sorted by the Data.
	HLS_SORTBYCODE	0x0004L	The list is sorted by the Codes.
	HLS_MULTISEL	0x0008L	Multiple items may be selected concurrently.
	HLS_BORDER3D	0x0010L	The list is drawn with a 3D border, either indented or extruded depending on the setting of the WS_BORDER flag.
	HLS_EXTRUDE	(HLS_BORDER3D   WS_BORDER)	The list is shown with a 3D extruded border.
	HLS_HILITE	0x0020L	The background color is changed to the HiliteColor when the control receives the input focus.
	HLS_USETABS	0x0080L	Tabs are expanded in text strings.
	HLS_NONINHEIGHT	0x0100L	A partial item can be displayed at the bottom of the list.
	HLS_MULTICOL	0x0200L	Items are wrapped in newspaper style columns; a horizontal scrollbar is used if



	HLS_EXTENDEDESEL	0x0800L	all the items do not fit in the window.
	HLS_HUNGRY	0x1000L	Multiple items may be selected concurrently; the mouse selects by dragging.
	HLS_NOScroll	0x4000L	The control swallows Enter and Esc keyboard messages and notifies its parent.
			Scrollbars are not added automatically when the list cannot be displayed in full







## HComb Styles

<b>Constant</b>	<b>Value</b>	<b>Description</b>
 HCS_BORDER3D	0x0010L	The list is drawn with a 3D border, either indented or extruded depending on the setting of the WS_BORDER flag.
 HCS_EXTRUDE	(HCS_BORDER3D   WS_BORDER)	The control is shown with a 3D extruded border.
 HCS_HILITE	0x0020L	The background color is painted with the HiliteBrush when the control receives the input focus.
 HCS_SORTBYCODE	0x0004L	The list is sorted by the Codes.
 HCS_SORTBYDATA	0x0002L	The list is sorted by the Data.
 HCS_DROPDOWN	0x0001L	The control displays its list in a drop-down box
 HCS_HASEDIT	0x0008L	The control accepts input in an edit box
 HCS_NONINHEIGHT	0x0100L	A partial item can be displayed at the bottom of the list.
 HCS_USETABS	0x0080L	Tabs are expanded in text strings.



## HGrid Styles

<b>Constant</b>	<b>Value</b>	<b>Description</b>
 HGS_BROWSE	0x0001L	The control will not allow editing.
 HGS_NOLINES	0x0002L	No grid lines will be displayed.
 HGS_MDICHILD	0x0004L	The control is an MDI Child window.
 HGS_INPLACE	0x0008L	Editing is done at the cell

<input type="checkbox"/>	HGS_AUTOEXTEND	0x0010L	location. New records are added to the bottom of the grid as the user scrolls down
<input type="checkbox"/>	HGS_RESIZEROWS	0x0020L	Rows are resizeable.
<input type="checkbox"/>	HGS_RESIZECOLS	0x0040L	Columns are resizeable
<input type="checkbox"/>	HGS_ROWBUTTONS	0x0080L	Numbered buttons appear at the beginning of each row.
<input type="checkbox"/>	HGS_COLBUTTONS	0x0100L	Buttons containing the field names appear at the top of each column.
<input type="checkbox"/>	HGS_KEYBDELINS	0x0200L	The keyboard Ins and Del keys can be used to insert and delete records.
<input type="checkbox"/>	HGS_LEAVEONTAB	0x0400L	Hitting the Tab key causes the grid to lose input focus
<input type="checkbox"/>	HGS_NOHIDSEL	0x0800L	The selection remains displayed when the control loses input focus
<input type="checkbox"/>	HGS_DRAGCOLS	0x1000L	Grid columns can be repositioned by the user dynamically
<input type="checkbox"/>	HGS_WHOLEROWS	0x2000L	Clicking on a cell selects the entire record
<input type="checkbox"/>	HGS_SINGLESELECT	0x4000L	Ranges of cells cannot be selected
<input type="checkbox"/>	HGS_DISABLENOSCROLL	0x8000L	Vertical scrollbar is disabled (not hidden) when it is not needed
<input type="checkbox"/>	WS_HSCROLL	0x00100000L	Horizontal scrollbar is shown when needed
<input type="checkbox"/>	WS_VSCROLL	0x00200000L	Vertical scrollbar is shown when needed

















## **HTool Styles**

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<input type="checkbox"/> HTS_BOTTOM	0x0001L	Creates a tool bar at the bottom of the parent window
<input type="checkbox"/> HTS_FLOAT	WS_POPUP	Creates a floating tool palette
<input type="checkbox"/> HTS_LEFT	0x0002L	Creates a tool along the left-hand side of the parent window
<input type="checkbox"/> HTS_NOSTRETCH	0x0010L	The tool bar is not automatically elongated
<input type="checkbox"/> HTS_RIGHT	0x0004L	Creates a tool along the right-hand side of the parent window
<input type="checkbox"/> HTS_TOP	0x0000L	Creates a tool bar at the top of the parent window

	HTS_RIBBON	(WS_CHILD   WS_VISIBLE)
	HTS_STATUS	(WS_CHILD   WS_VISIBLE   HTS_BOTTOM)
	HTS_PALETTE	(HTS_FLOAT   WS_CAPTION   WS_VISIBLE)



## HStat Styles

<b>Constant</b>	<b>Value</b>	<b>Description</b>
 HSS_BORDER3D	0x0010L	The control's border has a shaded,three-diminsional appearance
 HSS_BUMP	0x0020L	The control's border has a raised appearance
 HSS_EXTRUDE	(HSS_BORDER3D   WS_BORDER)	The control's border has an extruded appearance
 HSS_FRAME	0x0002L	The control appears as a frame
 HSS_GROUP	0x0001L	The control appears as a group box
 HSS_HLINE	0x0004L	The control appears as a horizontal line
 HSS_INDENT	(HSS_BORDER3D)	The control's border has an indented appearance
 HSS_LEFT	0x0040L	The control's text is left justified
 HSS_PIC	0x0003L	The control appears as a static bitmap or icon
 HSS_RIGHT	0x0080L	The control's text is right justified
 HSS_TEXT	0x0000L	The control appears as static text
 HSS_TEXTINDENT	0x0100L	The control's text is right indented
 HSS_TRANSPARENT	0x0200L	The button does not erase its background, allowing whatever is behind it to show through.
 HSS_VLINE	0x0005L	The control appears as vertical line

## HComb Messages

<b>Message</b>	<b>wParam</b>	<b>lParam</b>	<b>Return Value</b>
<u>HCM_BEQUIET</u>	bValue	0L	<i>Not Used</i>
<u>HCM_DELETEITEM</u>	iIndex	0L	bSuccess
<u>HCM_FINDCODE</u>	iStart	lpCode	iIndex
<u>HCM_FINDDATA</u>	iStart	lpData	iIndex
<u>HCM_FINDSTRING</u>	iStart	lpText	iIndex
<u>HCM_GETBKGNDBRUSH</u>	0	0L	hbrBkgnd

<u>HCM_GETCODE</u>	iIndex	lpCode	bSuccess
<u>HCM_GETCODECLASS</u>	0	0	cCodeClass
<u>HCM_GETCODELINK</u>	0	0L	lpCodeLink
<u>HCM_GETCODESIZE</u>	iIndex	0L	iCodeSize
<u>HCM_GETCODETYPE</u>	0	0L	cCodeType
<u>HCM_GETCOUNT</u>	0	0	iCount
<u>HCM_GETCURCODE</u>	wSize	lpBuf	iResult
<u>HCM_GETCURDATA</u>	wSize	lpBuf	iResult
<u>HCM_GETCURSEL</u>	0	0L	iCurSel
<u>HCM_GETDATA</u>	iIndex	lpData	bSuccess
<u>HCM_GETDATACLASS</u>	0	0L	cDataClass
<u>HCM_GETDATALINK</u>	0	0L	lpDataLink
<u>HCM_GETDATASIZE</u>	iIndex	0L	iDataSize
<u>HCM_GETDATATYPE</u>	0	0L	cDataType
<u>HCM_GETDROPHEIGHT</u>	0	0L	iDropHeight
<u>HCM_GETEDITDATA</u>	iMaxBytes	lpData	lBytesCopied
<u>HCM_GETEDITMAXTEXTLEN</u>	0	0L	iMaxLen
<u>HCM_GETEDITSCROLLPOS</u>	0	0L	iScrollPos
<u>HCM_GETEDITSEL</u>	0	0L	lSel
<u>HCM_GETEDITTEXT</u>	iMaxBytes	lpBuf	lBytesCopied
<u>HCM_GETEDITTEXTLEN</u>	0	0L	iEditTextLen
<u>HCM_GETFONT</u>	0	0L	hfFont
<u>HCM_GETFORMAT</u>	iMaxBytes	lpstrBuf	lBytesCopied
<u>HCM_GETHILITEBRUSH</u>	0	0L	hbrHilite
<u>HCM_GETOVERWRITEMODE</u>	0	0L	bOverwrite
<u>HCM_GETTEXT</u>	iIndex	lpBuf	lBytesCopied
<u>HCM_GETTEXTCOLOR</u>	bNegative	0L	crTextColor
<u>HCM_GETTEXTLEN</u>	iIndex	0L	iTextLen
<u>HCM_HASCHANGED</u>	0	0	bChanged
<u>HCM_ISQUIET</u>	0	0L	bQuiet
<u>HCM_RESETCONTENT</u>	0	0L	<i>Not Used</i>
<u>HCM_SELECTCODE</u>	wAction	lpCode	bSuccess
<u>HCM_SELECTDATA</u>	wAction	lpData	bSuccess
<u>HCM_SELECTSTRING</u>	iStart	lpText	bSuccess
<u>HCM_SETBKGNDBRUSH</u>	hbrNewBrush	0L	hbrOldBkgnd
<u>HCM_SETCHANGED</u>	bVal	0L	<i>Not Used</i>
<u>HCM_SETCODE</u>	iIndex	lpCode	bSuccess
<u>HCM_SETCODELINK</u>	bSelect	lpNewLink	lBytesCopied
<u>HCM_SETCURSEL</u>	iIndex	0L	iResult
<u>HCM_SETDATALINK</u>	bSelect	lpNewLink	lBytesCopied
<u>HCM_SETDROPHEIGHT</u>	iNewHeight	0L	bSuccess

<u>HCM_SETEDITDATA</u>	0	lpData	lBytesCopied
<u>HCM_SETEDITMAXTEXTLEN</u>	iLen	0L	<i>Not Used</i>
<u>HCM_SETEDITSCROLLPOS</u>	iScroll	bRedraw	iScrollPos
<u>HCM_SETEDITSEL</u>	0	lNewSel	<i>Not Used</i>
<u>HCM_SETFONT</u>	hfOldFont	hfNewFont	bRedraw
<u>HCM_SETFORMAT</u>	bRedraw	lpstrBuf	lBytesCopied
<u>HCM_SETHILITEBRUSH</u>	hbrNewHilite	0L	hbrOldHilite
<u>HCM_SETOVERWRITEMODE</u>	bMode	0L	<i>Not Used</i>
<u>HCM_SETTABSTOPS</u>	iNumber	lpTabs	bSuccess
<u>HCM_SETTEXTCOLOR</u>	bNegative	crNewColor	<i>Not Used</i>

## ■ HList Messages

<b>Message</b>	<b>wParam</b>	<b>lParam</b>	<b>Return Value</b>
<u>HLM_BEQUIET</u>	bVal	0L	<i>Not Used</i>
<u>HLM_DELETEITEM</u>	iIndex	0L	bSuccess
<u>HLM_FINDCODE</u>	iStart	lpCode	iIndex
<u>HLM_FINDDATA</u>	iStart	lpData	iIndex
<u>HLM_FINDSTRING</u>	iStart	lpText	iIndex
<u>HLM_GETBKGNDBRUSH</u>	0	0L	hbrBkgnd
<u>HLM_GETCODE</u>	iIndex	lpCode	bSuccess
<u>HLM_GETCODECLASS</u>	0	0L	cCodeClass
<u>HLM_GETCODELINK</u>	0	0L	lpCodeLink
<u>HLM_GETCODESIZE</u>	iIndex	0L	iCodeSize
<u>HLM_GETCODETYPE</u>	0	0L	cCodeType
<u>HLM_GETCOUNT</u>	0	0L	iCount
<u>HLM_GETCURCODE</u>	wSize	lpBuf	iResult
<u>HLM_GETCURDATA</u>	wSize	lpBuf	iResult
<u>HLM_GETCURSEL</u>	0	0L	iCurSel
<u>HLM_GETDATA</u>	iIndex	lpData	bSuccess
<u>HLM_GETDATACLASS</u>	0	0L	cDataClass
<u>HLM_GETDATALINK</u>	0	0L	lpDataLink
<u>HLM_GETDATASIZE</u>	0	0L	iDataSize
<u>HLM_GETDATATYPE</u>	0	0L	cDataType
<u>HLM_GETFONT</u>	0	0L	hfFont
<u>HLM_GETFORMAT</u>	iMaxBytes	lpstrBuf	lBytesCopied
<u>HLM_GETHILITEBRUSH</u>	0	0L	hbrHilite
<u>HLM_GETSEL</u>	iIndex	0L	bSelected
<u>HLM_GETSELCOUNT</u>	0	0L	iCount
<u>HLM_GETSELITEMS</u>	iMaxItems	lpBuf	iCopied
<u>HLM_GETSTATE</u>	0	0L	wState
<u>HLM_GETTEXT</u>	iIndex	lpBuf	lBytesCopied
<u>HLM_GETTEXTCOLOR</u>	bNegative	0L	crTextColor

<u>HLM_GETTEXTLEN</u>	iIndex	0L	iTextLen
<u>HLM_GETTOPINDEX</u>	0	0L	iTop
<u>HLM_HASCHANGED</u>	0	0L	bChanged
<u>HLM_ISQUIET</u>	0	0L	bQuiet
<u>HLM_RESETCONTENT</u>	0	0L	<i>Not Used</i>
<u>HLM_SELECTCODE</u>	wAction	lpCode	bSuccess
<u>HLM_SELECTDATA</u>	wAction	lpData	bSuccess
<u>HLM_SELECTITEM</u>	wAction	iIndex	bSuccess
<u>HLM_SELECTRANGE</u>	wAction	(iStart, iEnd)	iNumber
<u>HLM_SELECTSTRING</u>	iStart	lpText	bSuccess
<u>HLM_SETBKGNDBRUSH</u>	hbrNewBrush	0L	hbrOldBkgnd
<u>HLM_SETCODELINK</u>	bSelect	lpNewLink	lBytesCopied
<u>HLM_SETCOLUMNWIDTH</u>	iWidth	0L	bSuccess
<u>HLM_SETCURSEL</u>	iIndex	0L	iResult
<u>HLM_SETDATALINK</u>	bSelect	lpNewLink	lBytesCopied
<u>HLM_SETFONT</u>	hfNewFont	bRedraw	hfOldFont
<u>HLM_SETFORMAT</u>	bRedraw	lpstrBuf	lBytesCopied
<u>HLM_SETHILITEBRUSH</u>	hbrNewHilite	0L	hbrOldHilite
<u>HLM_SETREDRAW</u>	bSetting	bRedrawNow	<i>Not Used</i>
<u>HLM_SETSEL</u>	iAction	iIndex	bSuccess
<u>HLM_SETSTATE</u>	wFlag	bSetting	wNewState
<u>HLM_SETTABSTOPS</u>	iNumber	lpTabs	bSuccess
<u>HLM_SETTEXTCOLOR</u>	bNegative	crNewColor	<i>Not Used</i>
<u>HLM_SETTOPINDEX</u>	iIndex	0L	iTop

## ▣ HStat Messages

<b>Message</b>	<b>wParam</b>	<b>lParam</b>	<b>Return Value</b>
<u>HSM_GETBKGNDCOLOR</u>	0	0L	crBkgnd
<u>HSM_GETFRGNDCOLOR</u>	0	0L	crFrgnd
<u>HSM_GETPALETTE</u>	0	0L	hpPalette
<u>HSM_GETPIC</u>	0	0L	hPic
<u>HSM_GETTEXT</u>	iMaxBytes	lpBuf	lBytesCopied
<u>HSM_GETTYPE</u>	0	0L	wType
<u>HSM_SETBKGNDCOLOR</u>	0	crNewColor	<i>Not Used</i>
<u>HSM_SETFRGNDCOLOR</u>	0	crNewFrgnd	<i>Not Used</i>
<u>HSM_SETPIC</u>	0	(hPic, wType)	hOldPic
<u>HSM_SETTEXT</u>	0	lpBuf	<i>Not Used</i>

## ▣ HTool Messages

<b>Message</b>	<b>wParam</b>	<b>lParam</b>	<b>Return Value</b>
<u>HTM_BEQUIET</u>	bValue	0L	<i>Not Used</i>
<u>HTM_GETBRUSH</u>	0	0L	hbrBkgnd
<u>HTM_GETCAPTION</u>	wCount	lpBuf	<i>Not Used</i>
<u>HTM_GETNOTIFY</u>	0	0L	hwndNotify
<u>HTM_ISQUIET</u>	0	0L	bQuiet
<u>HTM_SETBRUSH</u>	hbrNewBrush	0L	hbrOldBkgnd
<u>HTM_SETCAPTION</u>	0	lpTitle	<i>Not Used</i>
<u>HTM_SETNOTIFY</u>	hwndNew	0L	<i>Not Used</i>

## ■ HGrid Messages

<b>Message</b>	<b>wParam</b>	<b>lParam</b>	<b>Return Value</b>
<u>HGM_ADDFLD</u>	0	hFld	iFld
<u>HGM_ADDREC</u>	0	lpRecData	iRowIndex
<u>HGM_BEQUIET</u>	bValue	0L	<i>Not Used</i>
<u>HGM_DELETEFLD</u>	iColIndex	0L	bSuccess
<u>HGM_DELETEREC</u>	iRowIndex	0L	bSuccess
<u>HGM_FINDFLD</u>	0	lpFldName	iColIndex
<u>HGM_GETBASEREC</u>	0	0L	lBaseRec
<u>HGM_GETBKGND BRUSH</u>	0	0L	hbrBkgnd
<u>HGM_GETBTNHEIGHT</u>	0	0L	wBtnHeight
<u>HGM_GETBTNWIDTH</u>	0	0L	iBtnWidth
<u>HGM_GETBUFFERPROC</u>	0	0L	lpfnBufferProc
<u>HGM_GETBUFFERSIZE</u>	0	0L	iBufSize
<u>HGM_GETCOLCOUNT</u>	0	0L	wColCount
<u>HGM_GETCOLMAP</u>	0	lpMap	<i>Not Used</i>
<u>HGM_GETCURREC</u>	0	lpRecData	bSuccess
<u>HGM_GETFIRSTCOL</u>	0	0L	wFirstCol
<u>HGM_GETFIRSTREC</u>	0	0L	lFirstRec
<u>HGM_GETFONT</u>	0	0L	hFont
<u>HGM_GETFROZENCOLS</u>	0	0L	wFrozenFlds
<u>HGM_GETMARKER</u>	0	0L	lMarker
<u>HGM_GETMAXREC</u>	0	0L	lMaxRec
<u>HGM_GETRECLINK</u>	0	0L	lpLink
<u>HGM_GETRECSIZE</u>	0	0L	wRecSize
<u>HGM_GETROWCOUNT</u>	0	0L	iRowCount
<u>HGM_GETROWHEIGHT</u>	0	0L	wRowHeight
<u>HGM_GETSELANCHOR</u>	0	lpCell	lCell
<u>HGM_GETSELEXTENT</u>	0	lpCell	lCell
<u>HGM_GETSTATE</u>	0	0L	lState
<u>HGM_GETTITLE</u>	wCount	lpBuf	<i>Not Used</i>
<u>HGM_INSERTFLD</u>	iAtIndex	hFld	iFld

<u>HGM_INSERTREC</u>	iAtRow	lpRecData	iRowIndex
<u>HGM_ISQUIET</u>	0	0L	bQuiet
<u>HGM_MOVECOL</u>	iFrom	iTo	iNewIndex
<u>HGM_MOVEROW</u>	iFrom	iTo	iRowIndex
<u>HGM_RESETCONTENT</u>	bRedraw	0L	<i>Not Used</i>
<u>HGM_SETBKGNDBRUSH</u>	hbrNewBrush	0L	hbrOldBkgnd
<u>HGM_SETBTNHEIGHT</u>	wNewHeight	bRedraw	<i>Not Used</i>
<u>HGM_SETBTNWIDTH</u>	wNewWidth	bRedraw	<i>Not Used</i>
<u>HGM_SETBUFFERPROC</u>	0	lpfnNewBufProc	lpfnOldBufProc
<u>HGM_SETBUFFERSIZE</u>	0	iNewBufSize	iBufSize
<u>HGM_SETCOLMAP</u>	0	lpMap	<i>Not Used</i>
<u>HGM_SETFIRSTCOL</u>	iColIndex	0L	<i>Not Used</i>
<u>HGM_SETFIRSTREC</u>	0	lNewFirstRec	lFirstRec
<u>HGM_SETFONT</u>	hfNewFont	bRedraw	hfOldFont
<u>HGM_SETFROZENCOLS</u>	wNumFrozen	0L	<i>Not Used</i>
<u>HGM_SETMARKER</u>	0	(iColIndex, iRowIndex)	<i>Not Used</i>
<u>HGM_SETMAXREC</u>	0	lNewMaxRec	bSuccess
<u>HGM_SETRECLINK</u>	0	lpBuf	bSuccess
<u>HGM_SETROWHEIGHT</u>	wNewHeight	bRedraw	<i>Not Used</i>
<u>HGM_SETSELANCHOR</u>	0	(iCol, iRow)	<i>Not Used</i>
<u>HGM_SETSELEXTENT</u>	bExtend	(iCol, iRow)	<i>Not Used</i>
<u>HGM_SETSTATE</u>	bValue	wStateFlag	<i>Not Used</i>
<u>HGM_SETTITLE</u>	0	lpTitle	<i>Not Used</i>
<u>HGM_UPDATE</u>	bErase	0L	<i>Not Used</i>

### **Record Messages**

<u>HGRM_GETDATA</u>	wRowIndex	lpRecStruct	bSuccess
<u>HGRM_GETSTATE</u>	wRowIndex	0L	wState
<u>HGRM_SETDATA</u>	wRowIndex	lpRecStruct	bSuccess
<u>HGRM_SETSTATE</u>	wRowIndex	(bValue, wStateFlag)	<i>Not Used</i>

### **Field Messages**

<u>HGFM_GETCODECLASS</u>	iFld	0L	cCodeClass
<u>HGFM_GETCODETYPE</u>	iFld	0L	cCodeType
<u>HGFM_GETCOLWIDTH</u>	iFld	0L	iWidth
<u>HGFM_GETCTLTYPE</u>	iFld	0L	cCtlType
<u>HGFM_GETDATACLASS</u>	iFld	0L	cFldDataClass
<u>HGFM_GETDATASIZE</u>	iFld	0L	iSize



<u>HGFM_GETDATATYPE</u>	iFld	0L	cDataType
<u>HGFM_GETDROPHEIGH</u>	iFld	0L	wHeight
<u>HGFM_GETFORMAT</u>	iFld	lpszFormat	lBytesCopied
<u>HGFM_GETFORMATLEN</u>	iFld	0L	wFormatLen
<u>HGFM_GETHCTL</u>	iFld	0L	hwControl
<u>HGFM_GETNAME</u>	iFld	lpszName	bSuccess
<u>HGFM_GETNAMELEN</u>	iFld	0L	iNameLen
<u>HGFM_GETOFFSET</u>	iFld	0L	iOffset
<u>HGFM_GETSTATE</u>	iFld	0L	wState
<u>HGFM_SETDROPHEIGHT</u>	iFld	wNewHeight	<i>Not Used</i>
<u>HGFM_SETNAME</u>	iFld	lpszName	bSuccess
<u>HGFM_SETSTATE</u>	iFld	(bValue, wStateFlag)	<i>Not Used</i>

## ▣ HComb Methods

```

int FAR PASCAL HCAddItemEx(HWND hwnd, LPVOID lpData, LPVOID lpCode);
int FAR PASCAL HCAddItems(HWND hwnd, int iCount, LPVOID lpData);
int FAR PASCAL HCAddItemsEx(HWND hwnd, int iCount, LPVOID lpData, LPVOID
lpCode);
BOOL FAR PASCAL HCDeleteItems(HWND hwnd, WORD wSearch, int iCount, LPVOID
lpInfo);
int FAR PASCAL HCGetItems(HWND hwnd, int iCount, WORD wReturn, LPVOID
lpReturn, WORD wSearch, LPVOID lpInfo);
int FAR PASCAL HCInsertItemEx(HWND hwnd, int iPos, LPVOID lpData, LPVOID
lpCode);
int FAR PASCAL HCInsertItems(HWND hwnd, int iPos, int iCount, LPVOID lpData);
int FAR PASCAL HCInsertItemsEx(HWND hwnd, int iPos, int iCount, LPVOID lpData,
LPVOID lpCode);

```

## ▣ HList Methods

int FAR PASCAL	HListAddItem(HWND hwnd, LPVOID lpData);
int FAR PASCAL	HListAddItemEx(HWND hwnd, LPVOID lpData, LPVOID lpCode);
int FAR PASCAL	HListAddItems(HWND hwnd, int iCount, LPVOID lpData);
int FAR PASCAL	HListAddItemsEx(HWND hwnd, int iCount, LPVOID lpData, LPVOID lpCode);
BOOL FAR PASCAL	HListDeleteItem(HWND hwnd, int iIndex);
BOOL FAR PASCAL	HListDeleteItems(HWND hwnd, WORD wSearch, int iCount, LPVOID lpInfo);
BOOL FAR PASCAL	HListEmptyList(HWND hwnd);
BOOL FAR PASCAL	HListGetCode(HWND hwnd, int iIndex, LPVOID lpCode);
BOOL FAR PASCAL	HListGetData(HWND hwnd, int iIndex, LPVOID lpData);
int FAR PASCAL	HListGetItems(HWND hwnd, int iCount, WORD wReturn, LPVOID lpReturn, WORD wSearch, LPVOID lpInfo);
int FAR PASCAL	HListInsertItem(HWND hwnd, int iPos, LPVOID lpData);
int FAR PASCAL	HListInsertItemEx(HWND hwnd, int iPos, LPVOID lpData, LPVOID lpCode);
int FAR PASCAL	HListInsertItems(HWND hwnd, int iPos, int iCount, LPVOID lpData);
int FAR PASCAL	HListInsertItemsEx(HWND hwnd, int iPos, int iCount, LPVOID lpData, LPVOID lpCode);
BOOL FAR PASCAL	HListSelectCode(HWND hwnd, LPVOID lpCode, WORD wAction);
BOOL FAR PASCAL	HListSelectData(HWND hwnd, LPVOID lpData, WORD wAction);
BOOL FAR PASCAL	HListSelectItem(HWND hwnd, int iIndex, WORD wAction);
BOOL FAR PASCAL	HListSelectItems(HWND hwnd, WORD wAction, WORD wSearch, int iCount, LPVOID lpInfo);
BOOL FAR PASCAL	HListSetCode(HWND hwnd, int iIndex, LPVOID lpCode);
BOOL FAR PASCAL	HListSetCodeLin(HWND hwnd, LPVOID lpCode, BOOL bSelect);
BOOL FAR PASCAL	HListSetDataLink(HWND hwnd, LPVOID lpData, BOOL bSelect);

## ▣ **HTool Methods**

HWND FAR PASCAL HToolCreate(HANDLE hInstance, LPCSTR lpTemplate, HWND  
hWndParent, HWND hWndNotify, DWORD dwStyle, WORD wID, int  
iXPos, int iYPos);

void FAR PASCAL HToolUpdate (HWND);

## ■ HGrid Methods

BOOL FAR PASCAL HGGetCellData (HWND hCtl, int iCol, int iRow, LPVOID lpDest);  
BOOL FAR PASCAL HGSetCellData (HWND hCtl, int iCol, int iRow, LPVOID lpSrc);  
BOOL FAR PASCAL HGSetCellString (HWND hCtl, int iCol, int iRow, LPSTR lpSrc);  
WORD FAR PASCAL HGGetCellText (HWND hCtl, int iCol, int iRow, LPSTR lpText, int iMax);  
BOOL FAR PASCAL HGridInvalidateRange(HWND hwnd, int iCol1, int iRow1, int iCol2, int iRow2);  
  
BOOL FAR PASCAL HGridInvalidateCell (HWND hwnd, int iCol, int iRow);  
LPSTR FAR PASCAL HGOffsetPtr (HWND hCtl, int iCol, LPVOID lpRec);  
HFLD FAR PASCAL HGFieldCreate(LPCSTR lpName, WORD wState, int iColWidth, int iDropHeight, char cDataClass, char cDataType, char cCodeClass, char cCodeType, int iSize, char cCtlType, long lCtlStyle, LPCSTR lpFmt);  
  
BOOL FAR PASCAL HGFieldDestroy (HFLD lpFld);



## **HButt Notify**

### **Notification Code**



HBN\_CLICKED



HBN\_DOUBLECLICKED

### **Meaning**

The user has clicked the button.

The user has double-clicked the button.



## **HComb Notify**

### **Notification Code**



HCN\_ERRSPACE



HCN\_SELCHANGE



HCN\_DBLCLK



HCN\_SETFOCUS



HCN\_KILLFOCUS

### **Meaning**

The control is unable to perform an operation because of memory constraints.

The listbox Selection has changed.

The user has double-clicked on an item.

The control has gained the input focus.

The control has lost the input focus.



## **HEdit Notify**

### **Notification Code**



HEN\_CHANGE



HEN\_ERRSPACE



HEN\_HSCROLL



HEN\_INVALID



HEN\_KILLFOCUS



HEN\_MAXTEXT



HEN\_SETFOCUS



HEN\_UPDATE

### **Meaning**

The user has changed the Text.

The control was unable to allocate a memory.

The user clicked on the horizontal scroll bar.

The user has entered an invalid date or other data item. Returning TRUE prevents focus from leaving the control.

The control has lost input focus.

The Text has reached TextMaxLen.

The control has gained input focus.

The control is about to display altered Text



## **HList Notify**

### **Notification Code**



HLN\_DBLCLK



HLN\_ERRSPACE



HLN\_KILLFOCUS



HLN\_SELCHANGE



HLN\_SETFOCUS

### **Meaning**

The user has double-clicked on an item.

The control is unable to perform an operation because of memory constraints.

The control has lost the input focus.

The listbox Selection has changed.

The control has gained the input focus.



## HGrid Notify

### **Notification Code**



HGN\_BOTTOM

### **Meaning**

The user has attempted to scroll off the bottom of the table.



HGN\_COLMOVED

The user has dragged a column to a new location



HGN\_COLSIZED

The user has resized a column



HGN\_DBLCLK

The user has double-clicked on a cell.



HGN\_DESTROY

The grid is about to be destroyed.



HGN\_ERRSPACE

The control is unable to perform an operation because of memory constraints.



HGN\_KILLFOCUS

The control has lost the input focus.



HGN\_RECCHANGED

The contents of a record have been changed.



HGN\_RECDELETE

A record is about to be deleted.



HGN\_RECNEW

A new record was inserted.



HGN\_RECSWITCH

The selection has been moved to a different record.



HGN\_ROWSIZED

The user has resized the rows.



HGN\_SELCHANGE

The Selection has changed.



HGN\_SELCHANGING

The selection is about to change. Changing the Marker changes the new selection.



HGN\_SELEXTENDING

The selection is about to extend. Changing the Marker changes the new selection.



HGN\_SETFOCUS

The control has gained the input focus.



HGN\_TOP

The user has attempted to scroll off the top of the table.

## C++ Quick Reference



### **Class Hierarchy**



### **Button Controls**



### **The ComboBox**



### **The Edit Control**



### **The Grid**



### **The List Control**



## **Static Controls**



### **The ToolBar**

Member Functions

Styles

Notification Codes

Window Text

Member Functions

Styles

Notification Codes

Window Text



Member Functions

Styles

Notification Codes

Window Text

Member Functions

Styles

Notification Codes

Window Text

Member Functions

Styles

Notification Codes

Window Text

Member Functions

Styles

Window Text

Member Functions  
Styles

## ■ CHButton, CHBCheck, CHB3State, CHBDefault, CHBRadio Classes

### Constructors

### Members

#### Defined in CHButton/THButton:

BOOL Attach(HWND hParent, int nID) ;  
BOOL Attach(HWND hWnd) ;  
int GetPAlign(void) ;  
CBrush \* GetBkgndBrush(void) ;  
int GetCount(void) ;  
CFont \* GetFont(void) ;  
COLORREF GetMaskColor(void) ;  
CPalette \* GetPalette(void) ;  
CGdiObject \* GetPic(int nIndex=0) ;  
int GetRAlign(void) ;  
HANDLE GetSound(void) ;  
int GetState(void) ;  
int GetStateCount(void) ;  
int GetTAlign(void) ;  
int GetText(LPSTR lpBuf, int iMaxBytes=-1) ;  
COLORREF GetTextColor(void) ;  
BOOL IsPressed(void) ;  
void Press(BOOL bPress) ;  
void SetPAlign(int iAlign) ;  
CBrush \* SetBkgndBrush(CBrush \*pBkgnd) ;  
CFont \* SetFont(CFont \*pFont, BOOL bRedraw=TRUE) ;  
void SetMaskColor(COLORREF clrMask) ;  
CGdiObject \* SetPic(CGdiObject \*pPic, WORD wType, int nIndex=0) ;  
void SetRAlign(int iAlign) ;  
HANDLE SetSound(HANDLE hSnd) ;  
int SetState(int iState, BOOL bRedraw = TRUE) ;  
void SetTAlign(int iAlign) ;  
void SetText(LPSTR lpszText) ;  
void SetTextColor(COLORREF clrText) ;

#### Defined in CHBCheck/THBCheck:

int GetData(LPVOID lpData) ;  
int GetDataSize(void) ;  
int SetData(LPVOID lpBuf) ;  
int SetDataLink(LPVOID lpBuf=NULL) ;

#### Defined in CHBRadio/THBRadio:

BOOL IsLastInGroup(void) ;

## ■ CHComb Class

### Constructors

### Members

int [AddItem](#)(LPVOID lpData);  
int [AddItemEx](#)(LPVOID lpData, LPVOID lpCode);  
int [AddItems](#)(int iCount, LPVOID lpData);  
int [AddItemsEx](#)(int iCount, LPVOID lpData, LPVOID lpCode);  
int [DeleteItem](#)(int iIndex);  
BOOL [DeleteItems](#)(WORD wSearch, int iCount, LPVOID lpInfo);  
BOOL [EmptyList](#)(void);  
int [FindCode](#)(int iStart, LPVOID lpCode);  
int [FindData](#)(int iStart, LPVOID lpData);  
CBrush \* [GetBkgndBrush](#)(void);  
BOOL [GetCode](#)(int iIndex, LPVOID lpCode);  
char [GetCodeClass](#)(void);  
LPVOID [GetCodeLink](#)(void);  
int [GetCodeSize](#)(int iIndex=-1);  
int [GetCodeType](#)(void);  
int [GetCount](#)(void);  
BOOL [GetCurCode](#)(LPVOID lpBuf);  
BOOL [GetCurData](#)(LPVOID lpBuf);  
int [GetCurSel](#)(void);  
BOOL [GetData](#)(int iIndex, LPVOID lpData);  
char [GetDataClass](#)(void);  
LPVOID [GetDataLink](#)(void);  
int [GetDataSize](#)(int iIndex=-1);  
int [GetDataType](#)(void);  
int [GetDropHeight](#)(void);  
int [GetEditData](#)(LPVOID lpData, int iMaxLen=-1);  
int [GetEditScrollPos](#)(void);  
LONG [GetEditSel](#)(void);  
int [GetEditText](#)(LPSTR lpText, int iMaxLen=-1);  
int [GetEditTextLen](#)(void);  
CFont \* [GetFont](#)(void);  
void [GetFormat](#)(LPSTR lpFormat, int iMaxLen=-1);  
CBrush \* [GetHiliteBrush](#)(void);  
int [GetItems](#)(int iCount, WORD wReturn, LPVOID lpReturn, WORD wSearch, LPVOID lpInfo);  
int [GetText](#)(LPSTR lpBuf, int iIndex);  
COLORREF [GetTextColor](#)(BOOL bNeg=FALSE);  
int [GetTextLen](#)(int iIndex);  
BOOL [HasChanged](#)(void);  
int [InsertItem](#)(int iPos, LPVOID lpData);  
int [InsertItemEx](#)(int iPos, LPVOID lpData, LPVOID lpCode);  
int [InsertItems](#)(int iPos, int iCount, LPVOID lpData);  
int [InsertItemsEx](#)(int iPos, int iCount, LPVOID lpData, LPVOID lpCode);

BOOL IsSelected(int iIndex);  
BOOL SelectCode(LPVOID lpCode);  
BOOL SelectData(LPVOID lpData);  
CBrush \* SetBkgndBrush(CBrush \*pBknd, BOOL bRedraw=TRUE);  
BOOL SetChanged(BOOL bChanged);  
BOOL SetCode(int iIndex, LPVOID lpCode);  
BOOL SetCodeLink(LPVOID lpCode);  
int SetCurSel(int iIndex);  
BOOL SetDataLink(LPVOID lpData);  
BOOL SetDropHeight(int iNewHeight);  
int SetEditData(LPVOID lpData);  
int SetEditScrollPos(int iScroll, BOOL bRedraw);  
void SetEditSel(int iAnchor, int iExtent);  
CFont \* SetFont(CFont \*pFont, BOOL bRedraw=TRUE);  
int SetFormat(LPSTR lpFormat, BOOL bRedraw=TRUE);  
CBrush \* SetHiliteBrush(CBrush \*pBrush, BOOL bRedraw=TRUE);  
void SetMaxTextLen(int iLen);  
void SetOverwrite(BOOL bOverwrite);  
void SetTextColor(COLORREF crNew, BOOL bNeg=FALSE);  
int SetTabStops(int iNum, LPINT lpTStops);



## ■ **CHEdit Class**

### **Members**

BOOL [BeQuiet](#)(BOOL bQuiet);  
void [Clear](#)();  
void [Copy](#)();  
void [Cut](#)();  
CBrush \* [GetBkgndBrush](#)(void);  
char [GetDataClass](#)(void);  
LONG [GetData](#)(LPVOID lpData, int iMaxBytes=-1);  
LONG [GetDataSize](#)(void);  
char [GetDataType](#)(void);  
CFont \* [GetFont](#)(void);  
LONG [GetFormat](#)(LPSTR lpszFormat, int iMaxBytes=-1);  
CBrush \* [GetHiliteBrush](#)(void); vt  
int [GetMaxTextLen](#)(void);  
BOOL [GetOverwriteMode](#)(void);  
char [GetPasswordChar](#)(void);  
LONG [GetScrollPos](#)(void);  
LONG [GetSel](#)(void);  
void [GetSelText](#)(LPSTR lpSelText, int iMaxBytes=-1);  
LONG [GetState](#)(void);  
COLORREF [GetTextColor](#)(BOOL bNeg = FALSE);  
LONG [GetText](#)(LPSTR lpBuf, int iMaxBytes=-1);  
LONG [GetTextLen](#)(void);  
VALIDATEPROC [GetValidate](#)(void);  
BOOL [HasChanged](#)(void);  
BOOL [IsQuiet](#)(void);  
void [Paste](#)();  
void [ReplaceSel](#)(LPSTR lpStr);  
CBrush \* [SetBkgndBrush](#)(CBrush \*pBkgnd);  
void [SetChanged](#)(BOOL bChanged);  
LONG [SetData](#)(LPVOID lpBuf);  
LONG [SetDataLink](#)(LPVOID lpBuf=NULL);  
CFont \* [SetFont](#)(CFont \*pFont, BOOL bRedraw=TRUE);  
LONG [SetFormat](#)(LPSTR lpString, BOOL bRedraw=TRUE);  
CBrush \* [SetHiliteBrush](#)(CBrush \*pHilite);  
void [SetMaxTextLen](#)(int iMaxLen);  
void [SetOverwriteMode](#)(BOOL bMode=TRUE);  
void [SetPasswordChar](#)(char c);  
LONG [SetScrollPos](#)(int iNumChars, BOOL bRedraw=TRUE);  
void [SetSel](#)(int iAnchor, int iExtent);  
BOOL [SetState](#)(LONG lStateFlag, BOOL bValue);  
BOOL [SetTextColor](#)(COLORREF cr, BOOL bNeg = FALSE);  
BOOL [SetText](#)(LPSTR lpBuf);  
void [SetValidate](#)(VALIDATEPROC lpfnValidate);  
void [Undo](#)();  
void [Update](#)(BOOL bParseText=FALSE, BOOL bUpdateText=TRUE);

```
int Validate(void);
```

## ■ CHGrid Class

### Members

void [AddFld](#)(HFLD& Fld);  
int [AddRec](#)(LPVOID IpRecStruct=NULL);  
void [BeQuiet](#)(BOOL bValue);  
int [ColToFld](#)(int c);  
void [DeleteFld](#)(int iColIndex);  
BOOL [DeleteRec](#)(int iRecIndex);  
HFLD [FieldCreate](#)(LPSTR IpName, WORD wState, int iWidth, int iHeight, char cDataClass, char cDataType, char cCodeClass, char cCodeType, int iSize, char cCtlType, DWORD ICtlStyle, LPCSTR IpFormat);  
  
int [FindFld](#)(LPSTR IpszFldName);  
int [FldToCol](#)(int f);  
LONG [GetBaseRec](#)(void);  
CBrush \* [GetBkgndBrush](#)(void);  
int [GetBtnHeight](#)(void);  
int [GetBtnWidth](#)(void);  
BUFFERPROC [GetBufferProc](#)(void);  
int [GetBufferSize](#)(void);  
BOOL [GetCellData](#)(int iField, int iRec, LPVOID IpData);  
WORD [GetCellText](#)(int iField, int iRec, LPSTR IpText, WORD wMax);  
char [GetCodeClass](#)(int iColIndex);  
char [GetCodeType](#)(int iColIndex);  
int [GetColCount](#)(void);  
void [GetColMap](#)(int FAR \*IpMap);  
int [GetColWidth](#)(int iColIndex);  
WORD [GetCtlType](#)(int iColIndex);  
void [GetCurRec](#)(LPVOID IpRecStruct);  
char [GetDataClass](#)(int iColIndex);  
BOOL [GetData](#)(int iRecIndex, LPVOID IpRecStruct);  
int [GetDataSize](#)(int iColIndex);  
char [GetDataType](#)(int iColIndex);  
int [GetDropHeight](#)(int iColIndex);  
WORD [GetFieldState](#)(int iColIndex);  
int [GetFirstCol](#)(void);  
LONG [GetFirstRec](#)(void);  
CFont \* [GetFont](#)(void);  
void [GetFormat](#)(int iColIndex, LPSTR IpszFormat);  
int [GetFormatLen](#)(int iColIndex);  
int [GetFrozenCols](#)(void);  
HWND [GetHCtl](#)(int iColIndex);  
LONG [GetMarker](#)(void);  
LONG [GetMaxRec](#)(void);  
void [GetName](#)(int iColIndex, LPSTR IpszName);  
int [GetNameLen](#)(int iColIndex);  
LPVOID [GetRecLink](#)(void);  
int [GetRecSize](#)(void);

WORD GetRecState(int iRecIndex);  
 int GetRowCount(void);  
 int GetRowHeight(void);  
 LONG GetSelAnchor(LPCELL lpCell=NULL);  
 LONG GetSelExtent(LPCELL lpCell=NULL);  
 LONG GetState(void);  
 void GetTitle(int iCount, LPSTR lpszTitle);  
 void InsertFld(int wBeforeFld, HFLD& fld);  
 int InsertRow(int iRecIndex, LPVOID lpRecStruct=NULL);  
 BOOL InvalidateCell(int iFld, int iRec);  
 BOOL InvalidateRange(int iFld1, int iRec1, int iFld2, int iRec2);  
 BOOL IsQuiet(void);  
 void MoveCol(int wOldIndex, int wNewIndex);  
 int MoveRow(int iOldIndex, int iNewIndex);  
 LPSTR OffsetPtr(int iFld, LPVOID lpRec);  
 int RecToRow(LONG rec);  
 void ResetContent(BOOL bRedraw = TRUE);  
 LONG RowToRec(int row);  
 CBrush \* SetBkgndBrush(CBrush \*pBrush, BOOL bRedraw=TRUE);  
 void SetBtnHeight(int iHeight, BOOL bRedraw=TRUE);  
 void SetBtnWidth(int iWidth, BOOL bRedraw=TRUE);  
 BUFFERPROC SetBufferProc(BUFFERPROC lpfnBufProc);  
 int SetBufferSize(int iSize);  
 BOOL SetCellData(int iField, int iRec, LPVOID lpData);  
 BOOL SetCellString(int iField, int iRec, LPSTR lpData);  
 void SetColMap(int FAR \*lpMap);  
 void SetColWidth(int iColIndex, int iNewWidth);  
 BOOL SetData(int iRecIndex, LPVOID lpRecStruct);  
 void SetDropHeight(int iColIndex, int iNewHeight);  
 WORD SetFieldState(int iColIndex, WORD wFlag, BOOL bValue);  
 void SetFirstCol(int iHeight, BOOL bRedraw=TRUE);  
 LONG SetFirstRec(int iRecIndex);  
 CFont \* SetFont(CFont \*pFont, BOOL bRedraw=TRUE);  
 void SetFrozenCols(int iFrznCols);  
 void SetMarker(int iColIndex, int iRowIndex);  
 LONG SetMaxRec(LONG lMax);  
 void SetName(int iColIndex, LPSTR lpszName);  
 void SetRecLink(LPVOID lpRecStruct);  
 void SetRecState(int iRecIndex, WORD wFlag, BOOL bValue);  
 void SetRowHeight(int iHeight, BOOL bRedraw=TRUE);  
 void SetSelAnchor(int iCol, int iRow);  
 void SetSelExtent(BOOL bExtend, int iCol, int iRow);  
 void SetState(BOOL bValue, WORD wFlag);  
 void SetTitle(int iCount, LPSTR lpszTitle);  
 void Update(BOOL bErase);

## CHList Class

### Members

int [AddItem](#)(LPVOID lpData);  
int [AddItemEx](#)(LPVOID lpData, LPVOID lpCode);  
int [AddItems](#)(int iCount, LPVOID lpData);  
int [AddItemsEx](#)(int iCount, LPVOID lpData, LPVOID lpCode);  
void [BeQuiet](#)(BOOL bQuiet);  
BOOL [Deleteltem](#)(int iIndex);  
int [Deleteltems](#)(WORD wSearch, int iCount, LPVOID lpInfo);  
int [FindCode](#)(int iStart, LPVOID lpCode);  
int [FindData](#)(int iStart, LPVOID lpData);  
int [FindString](#)(int iStart, LPSTR lpString);  
CBrush \* [GetBkgndBrush](#)(void);  
BOOL [GetCode](#)(int iIndex, LPVOID lpCode);  
char [GetCodeClass](#)(void);  
LPVOID [GetCodeLink](#)(void);  
int [GetCodeSize](#)(int iIndex=-1);  
int [GetCodeType](#)(void);  
BOOL [GetCurCode](#)(LPVOID lpBuf, WORD wSize=-1);  
BOOL [GetCurData](#)(LPVOID lpBuf, WORD wSize=-1);  
WORD [GetCurSel](#)(void);  
BOOL [GetData](#)(int iIndex, LPVOID lpData);  
char [GetDataClass](#)(void);  
LPVOID [GetDataLink](#)(void);  
int [GetDataSize](#)(int iIndex=-1);  
int [GetDataType](#)(void);  
CFont \* [GetFont](#)(void);  
void [GetFormat](#)(LPSTR lpBuf, int iMaxLen=-1);  
CBrush \* [GetHiliteBrush](#)(void);  
int [GetItems](#)(int iCount, WORD wReturn, LPVOID lpReturn, WORDwSearch, LPVOID lpInfo);  
BOOL [GetSel](#)(int iIndex);  
int [GetSelCount](#)(void);  
int [GetSelltems](#)(LPINT lpBuf, int iMaxItems);  
LONG [GetState](#)(void);  
int [GetText](#)(LPSTR lpBuf, int iIndex);  
COLORREF [GetTextColor](#)(BOOL bNeg=FALSE);  
int [GetTextLen](#)(int iIndex);  
int [GetTopIndex](#)(void);  
BOOL [HasChanged](#)(void);  
int [InsertItem](#)(int iPos, LPVOID lpData);  
int [InsertItemEx](#)(int iPos, LPVOID lpData, LPVOID lpCode);  
int [InsertItems](#)(int iPos, int iCount, LPVOID lpData);  
int [InsertItemsEx](#)(int iPos, int iCount, LPVOID lpData, LPVOID lpCode);  
BOOL [IsQuiet](#)(void);  
BOOL [SelectCode](#)(LPVOID lpCode, WORD wAction);  
BOOL [SelectData](#)(LPVOID lpData, WORD wAction);

BOOL SelectItem(int iIndex, WORD wAction);  
BOOL SelectItems(WORD wAction, WORD wSearch, int iCount, LPVOID lpInfo);  
int SelectString(int iStart, LPSTR lpString);  
CBrush \* SetBkgndBrush(CBrush \*pBknd, BOOL bRedraw=TRUE);  
BOOL SetChanged(BOOL bChanged);  
BOOL SetCode(int iIndex, LPVOID lpCode);  
BOOL SetCodeLink(LPVOID lpCode, BOOL bSelect=TRUE);  
BOOL SetColumnWidth(int iWidth);  
int SetCurSel(int iIndex);  
BOOL SetDataLink(LPVOID lpData, BOOL bSelect=TRUE);  
CFont \* SetFont(CFont \*pFont, BOOL bRedraw=TRUE);  
int SetFormat(LPSTR lpszFormat, BOOL bRedraw=TRUE);  
CBrush \* SetHiliteBrush(CBrush \*pBrush, BOOL bRedraw=TRUE) ;  
BOOL SetRedraw(BOOL bRedraw, BOOL bRedrawNow);  
BOOL SetSel(int iAction, int iIndex);  
LONG SetState(WORD wFlag, BOOL bSetting);  
BOOL SetTabStops(int iNum, LPINT lpStops);  
void SetTextColor(COLORREF crNew, BOOL bNeg=FALSE);  
int SetTopIndex(int iTop);

## ■ CHStat Class

### Members

```
COLORREF GetBkgndColor(void);
COLORREF GetFrgndColor(void);
CPalette * GetPalette(void);
CGdiObject * GetPic(void);
int GetText(LPSTR lpBuf, int iMaxLen=-1);
int GetType(void);
void SetBkgndColor(COLORREF cr, BOOL bRedraw);
void SetFrgndColor(COLORREF cr, BOOL bRedraw);
CGdiObject * SetPic(CGdiObject *Image, WORD wType, BOOL bRedraw=TRUE);
void SetText(LPSTR lpszText);
```

## ▣ **CHTool Class**

### **Members**

HBRUSH	<u>GetBrushHTool_Attr_Background</u> >main(void);
int	<u>GetCaption</u> (LPSTR lpBuf, int iMaxLen=-1);
HWND	<u>GetNotify</u> (void);
HBRUSH	<u>SetBrush</u> (HBRUSH hBkgnd);
void	<u>SetCaption</u> (LPSTR lpszCaption);
HWND	<u>SetNotify</u> (HWND hNotify);
void	<u>Update</u> ();



## ■ C++ Class Derivation Chart

### MFC Classes

<b>Class Name</b>	<b>Description</b>	<b>WinWidget Encapsulated</b>	<b>Derived From</b>
<u>CHButt</u>	Generic Pushbutton	HButt	CWnd
<u>CHBCheck</u>	CheckBox	HButt	CHButt
<u>CHB3State</u>	3 State Button	HButt	CHBCheck
<u>CHBDefault</u>	Defpushbutton	HButt	CHButt
<u>CHBRadio</u>	Radio Button	HButt	CHBCheck
<u>CHComb</u>	Combo Box	HComb	CWnd
<u>CHEdit</u>	Edit control	HEdit	CWnd
<u>CHGrid</u>	Grid control child	HGrid	CWnd
<u>CHGridView</u>	Grid control view	HGrid	CView
<u>CHGridBuffer</u>	Grid buffer object	N/A	CObject
<u>CHList</u>	List control	HList	CWnd
<u>CHStat</u>	Static control	HStat	CWnd
<u>CHTool</u>	Toolbar	HTool	CDialog

### OWL Classes

<b>Class Name</b>	<b>Description</b>	<b>WinWidget Encapsulated</b>	<b>Derived From</b>
<u>THButt</u>	Generic Pushbutton	HButt	TControl
<u>THBCheck</u>	CheckBox	HButt	THButt
<u>THB3State</u>	3 State Button	HButt	THBCheck
<u>THBDefault</u>	Defpushbutton	HButt	THButt
<u>THBRadio</u>	Radio Button	HButt	THBCheck
<u>THComb</u>	Combo Box	HComb	TControl
<u>THEdit</u>	Edit control	HEdit	TControl
<u>THGrid</u>	Grid control	HGrid	TControl
<u>THList</u>	List control	HList	TControl
<u>THStat</u>	Static control	HStat	TControl
<u>THTool</u>	Toolbar	HTool	TDialog

## Installing the WinWidgets

The **SETUP** program on the WinWidgets diskettes will help install your toolset.

In the SETUP program, use the **Set Path** button to enter a directory in which to install the WinWidgets. SETUP will create several sub-directories within this directory to hold various toolset components. One of these sub-directories, ...\`bin`, must be added to your PATH. If a previous version of the WinWidgets is present in the installation directory, the existing files will be over-written.

Select the toolset components you want to install by checking the boxes on the left side of the window. The WinWidgets' components and their meanings are listed below. Because components vary between editions, some may not be present in your package.

<b>Component</b>	<b>Content</b>
DLL and VBX files	The Dynamic Link Libraries containing the WinWidgets' code and the resource editor interface code.
On-line Manual	The WinWidgets help file, WIDGETS.HLP
Include/Declaration files	Header files containing definitions and declarations
Utilities	Utility programs, such as DaBoot
Libraries	Import libraries for linking to WIDGETS.DLL
Samples	Sample code for Visual Basic, C and C++ programmers.
OWL and MFC (.CPP) files	C++ source files containing OWL and MFC-compatible class wrappers for the WinWidgets
Source Code	Microsoft and Borland compatible C files and projects for rebuilding the WinWidgets

In the WinWidgets Professional Edition, SETUP can also help integrate the WinWidgets with the Microsoft Dialog Editor and/or Borland Resource Workshop. If you check the box labeled "Integrate with resource editors" SETUP will modify the initialization files for these tools to include the WinWidgets support DLL's. If you do not want SETUP to perform these modifications, make sure the box is not checked.

After installation, use the procedure described in [Resource Editor Integration](#) to integrate the WinWidgets with other programming tools.

For users of Microsoft Visual C++, SETUP can integrate the WinWidgets manual with the Visual Workbench Help System. This is a new and extremely useful feature that provides context-sensitive help for keywords defined in the WinWidgets manual. Once integrated, highlighting a word such as HEM\_SETFORMAT and pressing F1 will open the WinWidgets manual directly to the related topic.

To have SETUP integrate the manual, check both the **On-line Manual** and **Integrate with Visual Workbench** boxes. SETUP will execute the HELPINST.EXE utility, which modifies the MSVCHELP.IDX file in your \MSVC\HELP\ directory after making a backup called MSVCHELP.~SS. To undo the changes, copy the backup over the modified file.

To proceed with the installation, press **Install**. Installation can take a few minutes. Several files, such as the WinWidgets source code, are copied as self-extracting, compressed executables. If an error occurs during SETUP, you may extract the

contained files by simply calling the executable from DOS or Windows; the executable may then be deleted.

# **IMPORTANT - Read This Carefully Before Installation!**

## **Copyright and License Information**

**Copyright**           © 1992 Simple Software           Simple Software, Inc.  
All rights reserved.           543 3rd Street  
Brooklyn, NY 11215

**Copy and use restrictions**       The Software is protected by the copyright laws that pertain to computer software. Federal copyright law permits you to make one backup copy of the Software for your own use. It is illegal to duplicate the Software, by any means, in whole or part, for other purposes, except the individual files listed below, which may be distributed with applications that do not fall into the category of Application Design Tools (e.g. GUI builders, resource editors, etc.). Any other duplication requires a specific agreement with Simple Software.

WIDGETS.DLL

WIDGEVB.VBX

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means without the prior written consent of Simple Software. Information in this manual is subject to change without notice and does not represent a commitment on the part of the vendor.

**Disk warranty**           Simple Software warrants that the original diskettes are free from defects in material and workmanship, assuming normal use, for a period of ninety (90) days from date of purchase. You may return a defective disk within the given period to Simple Software, with a dated receipt; Simple Software will replace the disk free of charge. After 90 days, you may obtain a replacement by sending your faulty disk and check for \$20.00 to Simple Software.

EXCEPT FOR THE EXPRESS WARRANTY OF THE ORIGINAL DISKETTES SET FORTH ABOVE, SIMPLE SOFTWARE GRANTS NO OTHER WARRANTIES, EXPRESS OR IMPLIED, BY STATUTE OR OTHERWISE, REGARDING THE USE OF, OR THE RESULTS OF THE USE OF THIS SOFTWARE AND THE RELATED MATERIALS IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. THE LIABILITY OF SIMPLE SOFTWARE UNDER THE WARRANTY SET FORTH ABOVE SHALL BE LIMITED TO THE AMOUNT PAID BY THE CUSTOMER FOR THE PRODUCT. IN NO EVENT SHALL SIMPLE SOFTWARE BE LIABLE FOR ANY SPECIAL, CONSEQUENTIAL, OR OTHER DAMAGES FOR BREACH OF WARRANTY.

## Integration, Layout and Design

After installing the WinWidgets DLL's to your hard drive, you need to set up the Widgets as custom controls in your resource editor. The procedures listed here work for Microsoft's SDK Dialog Editor, AppStudio and Visual Basic, and Borland's Resource Workshop.


In addition to the procedures below, there are two important steps to using the WinWidgets controls in a C/C++ application. First, at the beginning of the application (WinMain() is a good place), the Widgets must be initialized. To do this, call

```
WidgetsInit();
```

Second, the WinWidgets header file, WIDGETS.H, must be included for both the compiler and resource compiler. In the Resource Workshop, you will receive "Expecting control window style" error messages if you do not include WIDGETS.H in a .RC project.

### Dialog Editor

In the Dialog Editor, select **File, Open Custom** and enter the path and name of the HBUTTON.DLL file. This adds the HButton control to the Dialog Editor's list of custom controls; in subsequent sessions the control will be loaded automatically. Repeat this procedure for each of the WinWidgets [support DLL's](#).

Once installed, the WinWidgets are instantiated by pressing the  button in the Toolbox. This will present a list of available custom controls. There may be a number of items in the list with the same name, each representative of a different control style. Choose the one closest to your needs and press OK. The Dialog Editor will present a rectangular cursor, allowing you to place the control in the dialog box.

Once placed, the control is ready to be customized. Select **Edit, Styles** or simply double-click on the control to bring up the Styles dialog. For a detailed description of each style flag, see the **Styles** section in the control's documentation.

### Borland's Resource Workshop

In Borland's Resource Workshop, you must create a DIALOG resource before installing the WinWidgets. Choose **File, New Project...**, select **.RC** and press **OK**. Then choose **Resource, New...**, select DIALOG and press **OK**. Next, choose **Options, Install Control Library...** and enter the path and name of the HBUTTON.DLL file. Repeat the last step for each of the WinWidgets [support DLL's](#). As the WinWidgets are integrated, they will appear as additional buttons in the Resource Workshop Tools palette.

The WinWidgets are instantiated by pressing the appropriate button in the Tools palette. The Resource Workshop will present a rectangular cursor, allowing you to place the control in the dialog box.

Once placed, the control is ready to be customized. Select **Control, Style** or simply double-click on the control to bring up the Styles dialog. For a detailed description of each style flag, see the **Styles** section in the control's documentation.

### Visual Basic

To install the WinWidgets in Visual Basic, open the AUTOLOAD.MAK project from the Visual Basic directory. Then choose **Add File...**, and select WIDGEVB.VBX from WIDGETS\BIN\ . The WinWidgets will appear as extra buttons in the Visual Basic Toolbox (choose **Window, Toolbox**) whenever a project is opened. The WinWidgets can also be installed for a single project by adding WIDGEVB.VBX to the project file rather than AUTOLOAD.MAK.

To create a control on a form, simply depress the appropriate button in the Toolbox, then create a rectangle for the control by clicking and dragging the mouse on the form. To edit the properties of a control, open the Properties window while the control is selected.


When distributing an application that uses the WinWidgets, include *both* WIDGEVB.VBX and WIDGETS.DLL on the distribution diskettes.

## AppStudio

In Microsoft's AppStudio, choose **Add File...**, and select WIDGEVB.VBX from WIDGETS\BIN\. The WinWidgets will appear as extra buttons in the Control Palette (choose **Window, Show Control Palette**) whenever a dialog resource is opened.

To create a VBX control in a form, simply depress the appropriate button in the Toolbox, then create a rectangle for the control by clicking and dragging the mouse on the dialog. To edit the properties of a control, open the Properties window (choose **Window, Show Properties**) while the control is selected, or double-click on the control.

When distributing an application that uses the WinWidgets as VBX controls, include *both* WIDGEVB.VBX and WIDGETS.DLL on the distribution diskettes.

It is possible  create an old style custom control by pressing the User Control button on the Toolbox.

. As with VBX controls, open the Properties window (choose **Window, Show Properties**) while the control is selected, or double-click on the control to edit the properties of a control. You must supply the appropriate window text (Caption), window class name (Class) and hexadecimal style bits (Style) in the Properties window. The C and C++ quick references of this manual will provide you with the necessary information to complete these fields. Non-VBX custom controls appear in AppStudio as featureless boxes and can only be tested in the compiled application, itself. Given the limited support for these types of controls in AppStudio, developers who do not wish to use VBX controls at all in their applications should consider designing dialogs in the SDK Dialog Editor or Borland's Resource Workshop instead of AppStudio.

When distributing an application that uses the WinWidgets as custom (non-VBX) controls, include only WIDGETS.DLL on the distribution diskettes.

Press this button in the Dialog Editor Toolbox to present a list of available custom controls.

Press this button in the AppStudio Toolbox to present a list of available custom controls.



## WinWidgets Files

The software provided with the WinWidgets toolset includes the Widgets DLL's, import library, and header, along with sample application code and executables. The files and their purposes are listed below. For those who bought the Widgets' source, a separate diskette contains the code and project files.

**These are the only files that may be redistributed with your application:**

WIDGETS.DLL	A library containing all of the WinWidgets controls.
WIDGEVB.VBX	A library containing the Visual Basic interface to the WinWidgets.

**As stated in the WinWidgets License Agreement, it is illegal to copy or distribute the files listed below:**

Help files:

WIDGETS.HLP	The WinWidgets on-line manual.
SETUP.EXE	A Windows-based installation program.
README.TXT	Contains information that was added after the completion of this manual.

Support DLL's:

HBUTT.DLL	The resource editor interface for the HButt button control.
HCOMB.DLL	The resource editor interface for the HComb combobox control.
HEDIT.DLL	The resource editor interface for the HEdit edit control.
HLIST.DLL	The resource editor interface for the HList list control.
HSTAT.DLL	The resource editor interface for the HStat static control.
HGRID.DLL	The resource editor interface for the HGrid grid control.
HCORE.DLL**	Core routines for the WinWidgets' resource editor interface.

**\*\* NOTE \*\*** Do not attempt to integrate HCORE.DLL or WIDGETS.DLL with a resource editor; they do not contain any code to interface with resource editors and may cause unexpected results.

Import library and declaration/include files:

WIDGETS.H	The include file for WIDGETS.DLL.
WIDGETS.LIB	The import library for WIDGETS.DLL.
WIDGEVB.TXT	Contains Visual Basic definitions and declarations.

Demo files:

SAMPLES\\*.\*

Source files and executables demonstrating the use and capabilities of the WinWidgets toolset.

