Table of Contents

InaGrid Overview

InaGrid

See Also FAQ Properties Methods Events Enums

InaGrid is an unbound/virtual 32-bit ActiveX grid control for use with VB, C++, Delphi or any language that supports OCX's. It is an ideal tool for database oriented viewers and editors where complex formatting is not required. It is also a good replacement for list boxes and list controls where you want to edit cells or display data fast. Because InaGrid is virtual, there are no pre-load or initialization delays; you can quickly open and scroll though very large files.

InaGrid gives you control over all aspects of a grid. Features include the following:

- Control color of background, foreground, regular or highlighted text, gridlines
- Horizontal and vertical alignment
- Selection by row or cell
- Multiple selection
- Over 999 billion rows by 2 billion columns
- Editing of any cell
- Moving and resizing of columns
- Multiple levels of column headers
- Owner-draw cells that can display bitmaps, etc.
- Creation of line styles
- Ability to embed an OCX in a cell

The InaGrid control is a system of objects:

- GridControl (contains the ColumnHeaders object)
- ColumnHeaders (contains the ColumnHeader object)
- ColumnHeader
- Format Data (used for cell formatting)
- DataObjectFiles (used for drag and drop)
- DataObject (used for drag and drop)

InaGrid also supplies three editing controls that can be used to capture user input:

- InaEdit
- InaCombo
- InaCheck

InaGrid Frequently Asked Questions (FAQ)

How do I create an InaGrid control in an application?

In Visual Basic 5.0:

To add the InaGrid control to your project, select Project | Components... Select the InaGrid ActiveX Control Module, InaEdit ActiveX Control Module, InaCombo ActiveX Control Module

and InaCheck ActiveX Control Module.

The InaGrid controls will now be available in the Visual Basic Toolbox. Select the InaGrid control from the Toolbox and insert it into a form as usual.

In Visual C++ 5.0:

To add the InaGrid control to your project, select Project | Add To Project | Components and Controls... In Registered ActiveX Controls folder select InaGrid Control. The InaGrid control wrapper class will be added to your project.

To insert the InaGrid control in an application window derived from CWnd, implement a WM CREATE message handler in your application window. Inside this function, create the InaGrid control as a child of your window by calling the Create method of the InaGrid control wrapper class.

To insert the InaGrid control in a dialog, choose InaGrid control from the Control toolbar and select the destination in the dialog where the control is to be placed. You can also click the right mouse button in the dialog. Choose Insert ActiveX control... from the popup-menu and then select InaGrid Control. The InaGrid control is inserted into your dialog.

How do I display row and column information?

The InaGrid control presents a virtual view on data in your program. Whereas most list and grid controls maintain a copy of the row and column information within their own internal data structures, the InaGrid control relies on the application to maintain its data – where it exists to begin with. Also, since the InaGrid control does not bind to a data source, it presents the programmer many more options to manipulate and check data before and after it is presented and edited in the control.

Because the InaGrid control presents a virtual view of your program data, it only requires that data be supplied by the application as it displays it onscreen. As the user scrolls through the data presented in the grid, the control asks the application for row and column information that will be needed for each page of data. Your application needs to be ready to respond to these requests as they occur.

The event that calls back into your application is ***GetData***. See the sample programs for examples of responding to this event and providing your application's data.

Using 64-bit data types

The large capacity of the InaGrid control is driven by the use of 64-bit numbers in all functions that require a row number. Visual Basic and Visual C++ handle 64-bit quantities differently. The InaGrid control provides a custom data type, ***ROWNUMBER***, that is equivalent to a ***CURRENCY*** data type in Visual Basic and a ***CY*** data type in Visual C++.

Visual Basic

The ***ROWNUMBER*** data type must be scaled properly in order to interpret the value. When using a value from a function that returns a ***ROWNUMBER***, you must multiply by 10,000. Conversely, to provide a scaled value to a function that expects a ***ROWNUMBER*** data type, you must divide by 10,000. When passing values to and from functions that expect a ***ROWNUMBER*** type, no scaling is necessary.

Visual C++

The ***ROWNUMBER*** data type in Visual C++ is equivalent to an ***CY*** data type. This requires all accesses to the variable to be through the ***int64*** union member.

How do I edit data in InaGrid cells?

By default, the InaGrid control is built to provide editing of an InaGrid cell when the user chooses to edit the cell data. The user initiates an edit action by double-clicking on the cell with the mouse, or pressing the F2 key. If an InaGrid cell has the focus, pressing ENTER sets the first editable cell on the row into edit mode.Subsequent presses of the TAB key moves through all of the editable cells in the row in turn, placing each one into edit mode.

If the user changes data by editing the cell, the InaGrid control informs the application of the changes using the ***SetData*** event. This provides your application with the new cell information for updating the application's information.

What kind of licensing agreement do I need to distribute InaGrid control?

The distribution of the InaGrid control is controlled by a license agreement. As a developer using the InaGrid control, you are required to purchase a license in order to distribute applications that contain one or more InaGrid controls. Without this license, the InaGrid control is considered under evaluation and displays a message box to indicate this. When you purchase a license, you are given a license file (**.lic*) that must be copied to the directory that the InaGrid control (*InaGrid.ocx*) resides in. You must not redistribute the license file.

If you are using Visual C++, you may define a variable that is passed as a parameter to the Create function. The contents of this variable contain the first line of text from the license file. Use the following method for defining and passing the license variable:

In Visual C++ 5.0:

```
static const CString strLicenseKey = "first line of text from license file";
BSTR bstrLicenseKey = strLicenseKey.AllocSysString();
BOOL bRet = m_wndGrid.Create(NULL, NULL, CRect(0, 0, 100, 100), this, 1, NULL,
FALSE, bstrLicenseKey);
SysFreeString(bstrLicenseKey);
```

Is there a way of selecting the whole row?

InaGrid provides a Row Select mode. Row selections can still be achieved in cell select mode (normally implemented by a left click in the Row Number area).

When InaGrid needs to highlight data (a selection was made), it uses an event called IsSelected. Like most InaGrid events, IsSelected passes the row, column and a flag for you to set that states whether this cell should be highlighted or not.

When I run the MFC samples I get a "Failed to create empty document" message and the application exits. Why?

InaGrid is not registered correctly. If you are evaluating InaGrid then you need to download the InaGridOCX Setup and install the InaGrid control.

GetData in an MFC application fails to add the OLE_COLOR pColor parameter correctly?

Make sure the lines in the .cpp file read:

```
ON_EVENT(object, 1, 1 /* OnGetData */, OnGetData, VTS_CY VTS_DISPATCH
VTS_PBSTR VTS_PCOLOR)
And
void object::OnGetData(CURRENCY nRow, LPDISPATCH pColumn, BSTR FAR* pValue,
OLE_COLOR* pColor)
{
}
```

And in .h file:

```
afx_msg void OnGetData(CURRENCY nRow, LPDISPATCH pColumn, BSTR FAR* pValue,
OLE_COLOR* pColor);
```

How do I contact Inabyte Inc.?

By Phone	415-883-3407 Sales 877-INA-BYTE
By Fax	415-898-1652 Sales and Support
By Email	support@inabyte.com info@inabyte.com
By Internet	http://www.inabyte.com/support.html http://www.inabyte.com
By US Mail	Inabyte Inc. 5 Betty Lane Novato, CA 94947 USA

Enums

```
GridColumnHAlignmentConstants
       gColumnLeft              = 0
       gColumnCenter            = 1
       gColumnRight             = 2
GridColumnVAlignmentConstants
       gColumnTop               = 0
       gColumnVCenter           = 4
       gColumnBottom            = 8
GridLineStyle
       gGridLineSolid           = 0
       gGridLineDot             = 1
       gGridLineDashDot         = 2
       gGridLineThinDash        = 3
       gGridLineWideDash        = 4
GridSelectConstants
       gSingleSelect            = 0
       gSelectRange             = 1
       gSelectToggle            = 2
GridSelectModeConstants
       gRowSelect               = 0
```

```
       gCellSelect              = 1
GridOLEDropConstants
       gOLEDropNone             = 0
       gOLEDropManual           = 1
GridDragOverConstants
       gEnter                   = 0
       gLeave                   = 1
       gOver                    = 2
GridOLEDropEffectConstants
       gOLEDropEffectNone       = 0
       gOLEDropEffectCopy       = 1
       GOLEDropEffectMove       = 2
       gOLEDropEffectLink       = 4
       gOLEDropEffectScroll     2147483648
```

InaGrid Control Objects

GridControl Object

GridControl Object

See Also FAQ Properties Methods Events Enums

The InaGrid control is a system of objects:

- GridControl (this object, which contains the ColumnHeaders object)
- ColumnHeaders (contains the ColumnHeader object)
- ColumnHeader
- Format Data (used for cell formatting)
- DataObjectFiles (used for drag and drop)
- DataObject (used for drag and drop)

Properties

Color Properties

See Also FAQ Properties Methods Events Enums

These are the color properties for the grid control:

- BackColor — returns or sets the background color of rows in an InaGrid control.
- ForeColor — returns or sets the foreground color used to display text in an InaGrid control.
- BackHiliteColor — returns or sets the background color used to display highlighted text in an InaGrid control.
- ForeHiliteColor — returns or sets the foreground color used to display highlighted text in an InaGrid control.
- GridLinesColor — returns or sets the color used to display lines separating rows and columns in an InaGrid control.

- ForeNumbersColor--returns or sets the foreground color used to display row numbers in an InaGrid control.

Syntax

VB

*Object.***BackColor** [= color]
*Object.***ForeColor** [= color]
*Object.***BackHiliteColor** [= color]
*Object.***ForeHiliteColor** [= color]
*Object.****GridLinesColor*** [= color]
*Object.***ForeNumbersColor** [= *color*]

C++

OLE_COLOR *Object.***GetBackColor** ()
void *Object.***SetBackColor** (OLE_COLOR color)
OLE_COLOR *Object.***GetForeColor** ()
void *Object.***SetForeColor** (OLE_COLOR color)
OLE_COLOR *Object.***GetBackHiliteColor** ()
void *Object.***SetBackHiliteColor** (OLE_COLOR color)
OLE_COLOR *Object.***GetForeHiliteColor** ()
void *Object.***SetForeHiliteColor** (OLE_COLOR color)
OLE_COLOR *Object.***GetGridLinesColor** ()
void *Object.***SetGridLinesColor** (OLE_COLOR color)
OLE_COLOR *Object.*GetForeNumbersColor ()
void *Object.***SetForeNumbersColor** (OLE_COLOR color)

The color property syntaxes have these parts:

Part	Description
object	An object expression that evaluates to an InaGrid control.
color	A value or constant that determines the background and foreground colors of an object, as described in Settings.

Settings

The InaGrid control uses the Microsoft Windows operating environment red-green-blue (RGB) color scheme. The settings for color are:

Setting	Description
Normal RGB colors	Colors specified by using the Color palette or by using the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VB) object library in the Object Browser . The Windows operating environment substitutes user choices as specified in the Control Panel settings.

The default settings at design time are as follows:

- BackColor — set to the system default color specified by the constant **vbWindowBackground.**
- ForeColor - set to the system default color specified by the constant **vbWindowText.**
- BackHiliteColor - set to the system default color specified by the constant **vb3DHighlight**.

- ForeHiliteColor - set to the system default color specified by the constant **vb3Dface.**
- GridLinesColor - set to the system default color specified by the constant **vb3Dshadow**.
- ForeNumbersColor - set to the system default color specified by the constant **vbWindowText**.

Remarks

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the Visual Basic (VB) object library in the Object Browser.

To display text in the Windows operating environment, both the text and background colors must be solid. If the text or background colors you've selected aren't displayed, one of the selected colors may be dithered--that is, comprised of up to three different-colored pixels. If you choose a dithered color for either the text or background, the nearest solid color will be substituted.

See Visual C++ Help. If MSB of color set, then color contains index used in GetSysColor: 0x800000xx, xx is a valid GetSysColor index.

Example

VB

```
GridControl1.BackColor = "&H00FF00"
GridControl1.ForeColor = "&H0000FF"
```

C++

```
m_wndGrid.SetForeColor(RGB(0x00, 0xff, 0x00));
m_wndGrid.SetBackColor(RGB(0x00, 0x00, 0xff));
```

ColumnHeaders Property

Returns a collection of ColumnHeader objects.

Syntax

VB

*object.***ColumnHeaders**

C++

CColumnHeaders *Object.***GetColumnHeaders**()

The object placeholder is an object expression that evaluates to an InaGrid control.

Remarks

The ColumnHeaders property returns a collection of ColumnHeader objects in a Variant.

You can manipulate many of an InaGrid control's attributes by changing the properties of ColumnHeader objects.

Example

VB

```
'Set focus to cell in second column
Dim aColHdrs As ColumnHeaders
Set aColHdrs = GridControl1.ColumnHeaders
GridControl1.FocusColumnHeader = aColHdrs.Item(1)
```

C++

```
// Set focus to cell in second column
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
CColumnHeader header = hdrCol.GetItem(1);
m_wndGrid.SetFocusColumnHeader(header);
```

ColumnHeadersIn3D Property

Returns or sets whether column headers and row numbers are displayed with 3D highlighting.

Syntax

VB

*Object.***ColumnHeadersIn3D** [= bValue]

C++

BOOL *Object.*GetColumnHeadersIn3D ()

void *Object.***SetColumnHeadersIn3D** (BOOL bValue)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
bValue	A Boolean expression specifying whether column headers and row numbers are displayed with 3D highlighting.

Settings

The settings for bValue are:

Setting	Description
True	Headers and row numbers are displayed with 3D highlighting.
False	Headers and row numbers are displayed with no highlighting and appear flat.

Example

VB

```
GridControl1.ColumnHeadersIn3D = False
```

C++

```
m_wndGrid.SetColumnHeadersIn3D(FALSE);
```

Container Property (Visual Basic Only)

Returns or sets the container of an InaGrid on a Form. Not available at design time.

Syntax

Set *Object.***Container** [= container]

The Container property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
container	An object expression that evaluates to an object that can serve as a container for other controls.

Count Property

Returns or sets the number of rows of an InaGrid control.

Syntax

VB

Object**.Count** [= number]

C++

CURRENCY *Object.***GetCount**()

*void Object.****SetCount****(CURRENCY& number)*

Part	Description
Object	An object expression that evaluates to an InaGrid control.
number	A 64-bit value or constant that determines the number of rows to display in an InaGrid control.

Remarks

VB:

The parameter **number** is a 64-bit value that is expressed in the Currency data type. To read this value properly you must multiply by 10,000. Conversely, to set this value, you must divide the row count by 10,000.

C++

The parameter **number** is a 64-bit value that is expressed in the CURRENCY data type. Use the ***int64*** union member to read and assign values using this type.

Example

VB

```
GridControl1.Count = 12 / 10000  'Set count to 12
```

C++

```
CURRENCY RowCount;
RowCount.int64 = 12;
m_wndGrid.SetCount(RowCount);
```

DeferUpdate Property

Returns or sets whether the InaGrid control displays updates to its appearance.

Syntax

VB

*Object.***DeferUpdate** [=bValue]

C++

BOOL *Object.***GetDeferUpdate**()

void *Object.***SetDeferUpdate** (BOOL bValue)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
bValue	A Boolean expression specifying whether updates are displayed.

Settings

The settings for Boolean are:

Setting	Description
True	All changes made to the control are immediately displayed.
False	Certain changes made to the control (font changes, count changes, row height changes) are recorded internally but are not displayed until the DeferUpdate property is set to TRUE or the control is repainted by another method.

Example

VB

```
GridControl1.DeferUpdate = True
```

C++

```
m_wndGrid.DeferUpdate(TRUE);
```

DragIcon Property (Visual Basic Only)

See Also FAQ Properties Methods Events Enums

Returns or sets the icon to be displayed as the pointer in a drag-and-drop operation.

Syntax

*Object.***DragIcon** [= icon]

The DragIcon property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
icon	Any code reference that returns a valid icon, such as a reference to a form's icon (Form1.Icon), a reference to another control's DragIcon property (Text1.DragIcon), or the LoadPicture function.

Settings

The settings for icon are:

Setting	Description
(none)	(Default) An arrow pointer inside a rectangle.
Icon	A custom mouse pointer. You specify the icon by setting it using the Properties window at design time. You can also use the LoadPicture function at run time. The file you load must have the .ico filename extension and format.

Remarks

You can use the DragIcon property to provide visual feedback during a drag-and-drop operation-- for example, to indicate that the source control is over an appropriate target. DragIcon takes effect when the user initiates a drag-and-drop operation. Typically, you set DragIcon as part of a MouseDown or DragOver event procedure.

Note At run time, the DragIcon property can be set to any object's DragIcon or Icon property, or you can assign it an icon returned by the LoadPicture function.

When you set the DragIcon property at run time by assigning the Picture property of one control to the DragIcon property of another control, the Picture property must contain an .ico file, not a .bmp file.

DragMode Property (Visual Basic Only)

See Also FAQ Properties Methods Events Enums

Returns or sets a value that determines whether manual or automatic drag mode is used for a

drag-and-drop operation.

Syntax

*Object.***DragMode** [= number]

The DragMode property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
number	An integer that specifies the drag mode, as described in Settings.

Settings

The settings for number are:

Constant	Setting	Description
vbManual	0	(Default) Manual--requires using the Drag method to initiate a drag-and-drop operation on the source control.
vbAutomatic	1	Automatic--clicking the source control automatically initiates a drag-and-drop operation. OLE container controls are automatically dragged only when they don't have the focus.

Remarks

When DragMode is set to 1 (Automatic), the control doesn't respond as usual to mouse events. Use the 0 (Manual) setting to determine when a drag-and-drop operation begins or ends; you can use this setting to initiate a drag-and-drop operation in response to a keyboard or menu command or to enable a source control to recognize a MouseDown event prior to a drag-and-drop operation.

Clicking while the mouse pointer is over a target object or form during a drag-and-drop operation generates a DragDrop event for the target object. This ends the drag-and-drop operation. A drag-and-drop operation may also generate a DragOver event.

Note: While a control is being dragged, it can't recognize other user-initiated mouse or keyboard events (KeyDown, KeyPress or KeyUp, MouseDown, MouseMove, or MouseUp). However, the control can receive events initiated by code or by a DDE link.

FocusColumnHeader Property

Returns or sets the column that is drawn with a focus rectangle.

Syntax

VB

*Object.***FocusColumnHeader** [= headerObject]

C++

CColumnHeader *Object.***GetFocusColumnHeader**()

void *Object.***SetFocusColumnHeader** (LPDISPATCH headerObject)

The FocusColumnHeader property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
headerObject	An object expression that evaluates to a ColumnHeader object.

Example

VB

```
'Set focus to cell in second column
Dim aColHdrs As ColumnHeaders
Set aColHdrs = GridControl1.ColumnHeaders
GridControl1.FocusColumnHeader = aColHdrs.Item(1)
```

C++

```
// Set focus to cell in second column
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
CColumnHeader header = hdrCol.GetItem(1);
m_wndGrid.SetFocusColumnHeader(header);
```

FocusRow Property

See Also FAQ Properties Methods Events Enums

Returns or sets the row that is drawn with a focus rectangle.

Syntax

VB

*Object.***FocusRow** [= number]

C++

CURRENCY *Object.***GetFocusRow**()

void *Object.***SetFocusRow** (CURRENCY& number)

The FocusRow property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
number	A numeric expression specifying the 0-based row that

contains the focus rectangle.

Remarks

The parameter **number** is a 64-bit value that is expressed in the CURRENCY data type. To read this value properly you must multiply by 10,000. Conversely, to set this value, you must divide the row count by 10,000.

If the value of **number** is –1, the focus is removed from all rows.

Example

VB

```
GridControl1.FocusRow = 0  / 10000 'set focus to first row
```

C++

```
CURRENCY FocusRow;
FocusRow.int64 = 0;
m_wndGrid.SetFocusRow(FocusRow);
```

Font Property

See Also FAQ Properties Methods Events Enums

Returns a Font object.

Syntax

VB

Object.**Font** [= *aFont*]

C++

COleFont *Object*.**GetFont**()

void *Object*.**SetFont**(LPDISPATCH aFont)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
aFont	A StdFont object defined in the Visual Basic StdType library.

Remarks

Use the Font property of an object to identify a specific Font object whose properties you want to use. For example, the following code changes the Bold property setting of a Font object identified by the Font property of an InaGrid control:

Example

VB

```
GridControl1.Font.Bold = True
```

C++

```
COleFont font = m_wndGrid.GetFont();
```

```
font.SetName(pDoc->m_lfText.lfFaceName)
```

GridLinesStyle Property

See Also FAQ Properties Methods Events Enums

Returns or sets the style of grid lines.

Syntax

VB

*Object.***GridLinesStyle** [= eGridLinesStyle]

C++

long *Object.*GetGridLinesStyle()

void *Object.***SetGridLinesStyle**(GridLinesStyle eGridLinesStyle)

Part	Description
Object	An object expression that evaluates to a ColumnHeader *Object.*
eGridLinesStyle	The type of line style to display grid lines.

Settings

The settings for eGridLinesStyle are:

Setting	Description
gGridLineSolid	Solid line.
gGridLineDot	Dotted line.
gGridLineDashDot	Dash-dotted line.
gGridLineThinDash	Thin line.
gGridLineWideDash	Wide line.

Remarks

The values for gGridLineSolid, gGridLineDot, gGridLineDashDot, gGridLineThinDash and gGridLineWideDash are respectively 0, 1, 2, 3 and 4.

Example

VB

```
GridControl1.GridLinesStyle = gGridLineDashDot
```

C++

```
CColumnHeader header = hdrCol.GetItem(1);
m_wndGrid.SetGridLinesStyle(gGridLineDashDot);
```

Height, Width Properties

See Also FAQ Properties Methods Events Enums

Returns or sets the dimensions of an InaGrid control.

Syntax

VB

*Object.***Height** [= number]

*Object.***Width** [= number]

C++

int *Object.***GetHeight**()

int *Object.***GetWidth**()

The Height and Width property syntaxes have these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
number	A numeric expression specifying the dimensions of an object, as described in Settings.

Settings

Measurements are calculated as follows:

Setting	Description
Number	Measured from the center of the control's border so that controls with different border widths align correctly. These properties use the scale units of a control's container.

Remarks

Width and height are read-only attributes; you may only get, not set.

The values for these properties change as the object is sized by the user or by your code. Maximum limits of these properties for all objects are system-dependent.

Use the Height, Width, Left, and Top properties for operations or calculations based on an object's total area, such as sizing or moving the object.

Example

VB

```
Dim iHeight As Integer
iHeight = GridControl1.Height
```

C++

```
int iGridHeight = m_wndGrid.GetHeight()
```

HelpContextID Property (Visual Basic Only)

Returns or sets an associated context number for an object. Used to provide context-sensitive

Help for your application.

Syntax

*Object.***HelpContextID** [= number]

The HelpContextID property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control. If object is omitted, the form associated with the active form module is assumed to be object.
number	A numeric expression that specifies the context number of the Help topic associated with object.

Settings

The settings for number are:

Setting	Description
0	(Default) No context number specified.
> 0	An integer specifying a valid context number.

Remarks

For context-sensitive Help on an object in your application, you must assign the same context number to both object and to the associated Help topic when you compile your Help file.

If you've created a Microsoft Windows operating environment Help file for your application and set the application's HelpFile property, when a user presses the F1 key, Visual Basic automatically calls Help and searches for the topic identified by the current context number.

The current context number is the value of HelpContextID for the object that has the focus. If HelpContextID is set to 0, then Visual Basic looks in the HelpContextID of the object's container, and then that object's container, and so on. If a nonzero current context number can't be found, the F1 key is ignored.

For a Menu control, HelpContextID is normally read/write at run time. But HelpContextID is read-only for menu items that are exposed or supplied by Visual Basic to add-ins, such as the Add-In Manager command on the Add-Ins menu.

Note Building a Help file requires the Microsoft Windows Help Compiler, which is included with the Visual Basic Professional Edition.

hWnd Property

Returns a handle to the InaGrid control.

Syntax

VB

*Object.***hWnd**

The object represents an object expression that evaluates to an InaGrid control.

C++

OLE_HANDLE *Object*.**GetHWnd**()

Part	Description
Object	An object expression that evaluates to an InaGrid control.
hWnd	A handle to the InaGrid control window.

Remarks

The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

This is read-only property.

Note: Because the value of this property can change while a program is running, never store the hWnd value in a variable.

Example

VB

```
nRet = OSWinHelp(GridControl1.hWnd, App.HelpFile, 3, 0)
```

C++

```
SetFocus((HWND)m_wndGrid.GetHWnd());
```

Index Property (Visual Basic Only)

Returns or sets the number that uniquely identifies an object in a collection.

Syntax

Object.**Index**

The object placeholder is an object expression that evaluates to an InaGrid control.

Remarks

The Index property is set by default to the order of the creation of objects in a collection. The index for the first object in a collection will always be one.

The value of the Index property of an object can change when objects in the collection are reordered, such as when you set the Sorted property to True. If you expect the Index property to change dynamically, it may be more useful to refer to objects in a collection by using the Key property.

Left, Top Properties (Visual Basic Only)

- Left--returns or sets the distance between the internal left edge of the InaGrid control and the left edge of its container.
- Top--returns or sets the distance between the internal top edge of the InaGrid control and the top edge of its container.

Syntax

*Object.***Left** [= value]

*Object.***Top** [= value]

The Left and Top property syntaxes have these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
value	A numeric expression specifying distance.

Remarks

The Left and Top properties are measured in units depending on the coordinate system of its container. The values for these properties change as the object is moved by the user or by code.

For either property, you can specify a single-precision number.

Use the Left, Top, Height, and Width properties for operations based on an object's external dimensions, such as moving or resizing.

MovableColumnHeaders Property

Returns or sets the ability to move column headers.

Syntax

VB

*Object.***MovableColumnHeaders** [= bValue]

C++

BOOL *Object.***GetMovableColumnHeaders()**

void *Object.***SetMovableColumnHeaders**(BOOL bMove)

Part	Description

Object	An object expression that evaluates to a GridControl object.
bValue	A boolean value

Example

VB

```
GridControl.MovableColumnHeaders = False
```

C++

```
pGridControl->SetMovableColumnHeaders(FALSE);
```

MultipleSelection Property

See Also FAQ Properties Methods Events Enums

Returns or sets whether multiple rows or cells can be selected in an InaGrid control.

Syntax

VB

*Object.*MultipleSelection [= bValue]

C++

BOOL *Object.*GetMultipleSelection()

void *Object.***SetMultipleSelection** (BOOL bValue)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
bValue	A Boolean expression specifying whether multiple selections can be made.

Settings

The settings for bValue are:

Setting	Description
True	Multiple selections can be made using Windows 95 conventions.
False	Only a single selection can exist.

Remarks

If the MultipleSelection property is true and the InaGrid control is in row selection mode (see SelectMode), multiple rows can be selected. Otherwise, if the control is in cell selection mode, multiple cells can be selected.

Example

VB

```
Dim bMultiSelect As Boolean
bMultiSelect = GridControl1.MultipleSelection
```

C++

```
BOOL bMultiSelect = m_wndGrid.GetMultipleSelection();
```

Name Property (Visual Basic Only)

Returns the name used in code to identify the InaGrid control object. Read-only at run time.

Syntax

Object.**Name**

An object expression that evaluates to an InaGrid control. If object is omitted, the form associated with the active form module is assumed to be object.

Remarks

The default name for new objects is the kind of object plus a unique integer. For example, the first new InaGrid control is GridControl1.

An object's Name property must start with a letter and can be a maximum of 40 characters. It can include numbers and underline (_) characters but can't include punctuation or spaces. Although the Name property setting can be a keyword, property name, or the name of another object, this can create conflicts in your code.

You can create an array of InaGrid controls by setting the Name property to the same value. For example, when you set the name of all InaGrid controls in a group to MyGrid, Visual Basic assigns unique values to the Index property of each control to distinguish it from others in the array. Two controls of different types can't share the same name.

Object Property (Visual Basic Only)

Returns the object and/or a setting of an InaGrid control method or property.

Syntax

*Object.**Object***[.property | .method]

The Object property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
property	Property that the InaGrid control object supports.
method	Method that the InaGrid control object supports.

Remarks

Use this property to specify an InaGrid control object you want to use in an Automation task.

You use the object returned by the Object property in an Automation task by using the properties and methods of the InaGrid control object.

OLEDropMode Property

Returns or sets whether the component or the programmer handles an OLE drag/drop operation.

Syntax

VB

*Object.***OLEDropMode** = mode

C++

*Object.***OLEDropMode** = mode;

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
mode	An integer which specifies the method with which an component handles OLE drag/drop operations, as described in Settings.

Settings

The settings for mode are:

Constant	Value	Description
GOLEDropManual	0	(Default) Manual. The programmer handles all OLE drag/drop operations.
gOLEDropAutomatic	1	Automatic. The component handles all OLE drag/drop operations.

Example

VB

```
OLEDropMode     =   1  'Manual
```

C++

```
m_pGrid->OLEDropMode = gOLEDropManual;
```

Remarks

When OLEDragMode is set to Manual, you must call the OLEDrag method to start dragging, which then triggers the OLEStartDrag event.

When OLEDragMode is set to Automatic, the source component fills the DataObject object with

the data it contains and sets the effects parameter before initiating the OLEStartDrag event (as well as the OLESetData and other source-level OLE drag/drop events) when the user attempts to drag out of the control. This gives you control over the drag/drop operation and allows you to intercede by adding other formats, or by overriding or disabling the automatic data and formats using the Clear or SetData methods.

If the source's OLEDragMode property is set to Automatic, and no data is loaded in the OLEStartDrag event, or aftereffects is set to 0, then the OLE drag/drop operation does not occur.

Note If the DragMode property of a control is set to Automatic, the setting of OLEDragMode is ignored, because regular Visual Basic drag and drop events take precedence.

Parent Property (Visual Basic Only)

Returns the form, object, or collection that contains the InaGrid control.

Syntax

*Object.***Parent**

The object placeholder is an object expression that evaluates to an InaGrid control.

Remarks

Use the Parent property to access the properties, methods, or controls of an InaGrid control's parent. For example:

```
GridControl1.Parent.MousePointer = 4
```

The Parent property is useful in an application in which you pass objects as arguments. For example, you could pass a control variable to a general procedure in a module, and use the Parent property to access its parent form.

There is no relationship between the Parent property and the MDIChild property. There is, however, a parent-child relationship between an MDIForm object and any Form object that has its MDIChild property set to True.

RowHeight Property

Returns or sets the height of a row of an InaGrid control.

Syntax

VB

*Object.***RowHeight** [= number]

C++

long *Object.***GetRowHeight**()

void *Object.***SetRowHeight** (long number)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
Number	A value or constant that determines the height in pixels of a row in an InaGrid control. If the value is negative the row height is calculated automatically based on current Font Height and Text Gutter.

Remarks

If RowHeight is positive, Height is set by the user. If RowHeight is the current auto line height, it is calculated from the font size.

Example

VB

```
GridControl1.RowHeight = 25
```

C++

```
m_wndGrid.SetRowHeight(25);
```

ScalePrint Property

Returns or sets whether InaGrid control data is scaled to print on a single page.

Syntax

VB

*Object.***ScalePrint** [= bValue]

C++

BOOL *Object.***GetScalePrint**()

void *Object.***SetScalePrint** (BOOL bValue)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
bValue	A Boolean expression specifying whether control data is scaled.

Settings

The settings for bValue are:

Setting	Description
True	All of the InaGrid data is scaled to print on a single page.
False	The InaGrid data prints normally according to the printer

capabilities.

Example

VB

```
GridControl1.ScalePrint = True
```

C++

```
m_wndGrid.SetScalePrint(TRUE);
```

SelectMode Property

See Also FAQ Properties Methods Events Enums

Returns or sets the selection behavior of the InaGrid control.

Syntax

VB

*Object.***SelectMode** [=*lValue*]

C++

long *Object.***GetSelectMode**()

void *Object.***SetSelectMode** (long lValue)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
lValue	A number or constant that specifies the selection mode of the control.

Settings

The settings for *lValue* are:

Setting	Description
gCellSelect	Selections are made on individual cells within the control.
gRowSelect	Selections made on any cell within a row select the row.

Example

VB

```
GridControl1.SelectMode = gRowSelect
```

C++

```
m_wndGrid.SetSelectMode(gRowSelect);
```

ShowGridLines Property

Returns or sets whether grid lines are displayed in the rows and columns of an InaGrid control.

Syntax

VB

*Object.***ShowGridLines** [= bValue]

C++

BOOL *Object.*GetShowGridLines()

void *Object.***SetShowGridLines** (BOOL bValue)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
bValue	A Boolean expression specifying whether grid lines are displayed.

Settings

The settings for bValue are:

Setting	Description
True	Grid lines are displayed in the rows and columns of the control using the GridLinesColor property.
False	No gridlines are displayed within the rows and columns of the control.

Example

VB

```
GridControl1.ShowGridLines = False
```

C++

```
m_wndGrid.SetShowGridLines(FALSE);
```

ShowHeaders Property

Returns or sets whether the InaGrid control displays headers and sub-headers.

Syntax

VB

*Object.***ShowHeaders** [= boolean]

C++

BOOL *Object.***GetShowHeaders**()

void *Object.***SetShowHeaders** (BOOL bValue)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
Boolean	A Boolean expression specifying whether headers and sub-headers are displayed.

Settings

The settings for bValue are:

Setting	Description
True	Column headers and any sub-headers that are defined are displayed.
False	No headers or sub-headers are displayed.

Example

VB

```
GridControl1.ShowHeaders = True
```

C++

```
m_wndGrid.SetShowHeaders(TRUE);
```

ShowNumbers Property

See Also FAQ Properties Methods Events Enums

Returns or sets whether row numbers are displayed in an InaGrid control.

Syntax

VB

*Object.***ShowNumbers** [= bValue]

C++

BOOL *Object.***GetShowNumbers**()

void *Object.*SetShowNumbers(BOOL bValue)

Part	Description
object	An object expression that evaluates to an InaGrid control.
bValue	A Boolean expression specifying whether row numbers are displayed for each row in the control.

Settings

The settings for Boolean are:

Setting	Description

True	An additional column exists on the left side of the control that can contain any program-provided data such as the line number of each row.
False	The line number column is not present.

Remarks

Use this property to provide an extra column that can display row number information. The programmer is responsible for providing data for this column using the *GetData* event handler. No ColumnHeader object for this column exists.

Example

VB

```
GridControl1.ShowNumbers = True
```

C++

```
m_wndGrid.SetShowNumbers(TRUE);
```

TabIndex Property (Visual Basic Only)

Returns or sets the tab order of the InaGrid control object within its parent form.

Syntax

*Object.***TabIndex** [= index]

The TabIndex property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
index	An integer from 0 to (n–1), where n is the number of controls on the form that have a TabIndex property. Assigning a TabIndex value of less than 0 generates an error.

Remarks

By default, Visual Basic assigns a tab order to controls as you draw them on a form, with the exception of the Menu, Timer, Data, Image, Line and Shape controls, which are not included in the tab order. At run time, invisible or disabled controls and controls that can't receive the focus (Frame and Label controls) remain in the tab order but are skipped during tabbing.

Each new control is placed last in the tab order. If you change the value of a control's TabIndex property to adjust the default tab order, Visual Basic automatically renumbers the TabIndex of other controls to reflect insertions and deletions. You can make changes at design time using the Properties window or at run time in code.

The TabIndex property isn't affected by the ZOrder method.

Note A control's tab order doesn't affect its associated access key. If you press the access key for a Frame or Label control, the focus moves to the next control in the tab order that can receive the focus.

When loading forms saved as ASCII text, controls with a TabIndex property that aren't listed in the form description are automatically assigned a TabIndex value. In subsequently loaded controls, if existing TabIndex values conflict with earlier assigned values, the controls are automatically assigned new values.

When you delete one or more controls, you can use the Undo command to restore the controls and all their properties except for the TabIndex property, which can't be restored. TabIndex is reset to the end of the tab order when you use Undo.

TabStop Property (Visual Basic Only)

Returns or sets a value indicating whether a user can use the TAB key to give the focus to an InaGrid control object.

Syntax

*Object.***TabStop** [= boolean]

The TabStop property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
boolean	A Boolean expression specifying whether the object is a tab stop, as described in Settings.

Settings

The settings for boolean are:

Setting	Description
True	(Default) Designates the InaGrid control as a tab stop.
False	Bypasses the InaGrid control when the user is tabbing, although the control still holds its place in the actual tab order, as determined by the TabIndex property.

Remarks

This property enables you to add or remove an InaGrid control from the tab order on a form.

Tag Property (Visual Basic Only)

Returns or sets an expression that stores any extra data needed for your program. Unlike other properties, the value of the Tag property isn't used by Visual Basic; you can use this property to identify objects.

Syntax

*Object.***Tag** [= expression]

The Tag property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
expression	A string expression identifying the InaGrid control. The default is a zero-length string ("").

Remarks

You can use this property to assign an identification string to an InaGrid control without affecting any of its other property settings or causing side effects. The Tag property is useful when you need to check the identity of an InaGrid control that is passed as a variable to a procedure.

TextGutter Property

Returns or sets the size of the margin that surrounds the text within rows and columns of an InaGrid control.

Syntax

VB

*Object.***TextGutter** [=lValue]

C++

long *Object.***GetTextGutter**()

void *Object.***SetTextGutter** (long lValue)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
lValue	A value or constant in pixels that determines the size of the margin surrounding text.

Example

VB

```
GridControl1.TextGutter = 3
```

C++

```
m_wndGrid.SetTextGutter(3);
```

ToolTipText Property (Visual Basic Only)

Returns or sets a ToolTip.

Syntax

*Object.***ToolTipText** [= string]

The ToolTipText property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
string	A string, associated with an InaGrid control, that appears in a small rectangle when the user's cursor hovers over the control at run time for about one second.

Remarks

At design time you can set the ToolTipText property string in the control's properties dialog box.

VerticalOffset, HorizontalOffset Properties

These are the offset properties:

- VerticalOffset - returns or sets the offset of the first visible row of the control.
- HorizontalOffset - returns or sets the horizontal offset of the grid

Syntax

VB

*Object.***VerticalOffset** [= number]

*Object.***HorizontalOffset** [= number]

C++

CURRENCY *Object.***GetVerticalOffset**()

void *Object.***SetVerticalOffset** (CURRENCY& number)

CY *Object.*GetHorizontalOffset()

void *Object.***SetHorizontalOffset** (CY& number)

The VerticalOffset and HorizontalOffset property syntaxes have these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid

	control.
number	A value or constant that determines the 0-based offset. The vertical offset determines the first row that is displayed at the top of the control. The horizontal offset determines the pixel offset of left side of the control.

Remarks

The parameter **number** is a 64-bit value that is expressed in the CURRENCY data type. To read this value properly you must multiply by 10,000. Conversely, to set this value, you must divide the row count by 10,000.

When setting either of the offsets, the control automatically scrolls if necessary. The current selection is not affected by the vertical offset.

The InaGrid control will always attempt to fill its current Height and Width. For this reason you cannot set a vertical offset value that is larger than the number of rows that can be displayed within the control, nor a horizontal offset that would cause the grid to scroll beyond the right-hand side of the control.

Example

VB

```
GridControl1.VerticalOffset = 100  / 10000 'scroll to row 101
```

C++

```
CURRENCY Offset;
Offset.int64 = 100;
m_wndGrid.SetVerticalOffset(Offset);
```

Visible Property (Visual Basic Only)

Returns or sets a value indicating whether the InaGrid control is visible or hidden.

Syntax

*Object.***Visible** [= boolean]

The Visible property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
boolean	A Boolean expression specifying whether the InaGrid control is visible or hidden.

Settings

The settings for boolean are:

Setting	Description
True	(Default) The control is visible.
False	The control is hidden.

Remarks

To hide the InaGrid control at startup, set the Visible property to False at design time. Setting this property in code enables you to hide and later redisplay the control at run time in response to a particular event.

WhatsThisHelpID Property (Visual Basic Only)

Returns or sets an associated context number for an InaGrid control. Use to provide context-sensitive Help for your application using the What's This pop-up in Windows 95 Help.

Syntax

*Object.***WhatsThisHelpID** [= number]

The WhatsThisHelpID property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
number	A numeric expression specifying a Help context number, as described in Settings.

Settings

The settings for number are:

Setting	Description
0	(Default) No context number specified.
>0	An integer specifying the valid context number for the What's This topic associated with the InaGrid control.

Remarks

Windows 95 uses the What's This button in the upper-right corner of the window to start Windows Help and load a topic identified by the WhatsThisHelpID property.

WrapColumnHeaders Property

Returns or sets whether column header text is word-wrapped in an InaGrid control.

Syntax

VB

*Object.***WrapColumnHeaders** [= bValue]

C++

BOOL *Object.***GetWrapColumnHeaders**()

void *Object.***SetWrapColumnHeaders** (BOOL bValue)

Part	Description
Object	An object expression that evaluates to an InaGrid control.
bValue	A Boolean expression specifying whether heading text is word-wrapped.

Settings

The settings for bValue are:

Setting	Description
True	Column header text wraps to the next line, and the height of the header row is increased accordingly, if the width of the column is not sufficient to display the header text string.
False	No word-wrapping occurs if the column width is insufficient to display the full text. An ellipsis is displayed in place of the remaining text.

Remarks

This property cannot be combined with vertical alignment of ColumnHeader objects.

Example

VB

```
GridControl1.WrapColumnHeaders = True
```

C++

```
m_wndGrid.SetWrapColumnHeaders(TRUE);
```

Methods

About Method

Displays the About box for the InaGrid control.

Syntax

VB

*object.***AboutBox**

C++

void *Object.***AboutBox**()

The object placeholder is an object expression that evaluates to an InaGrid control.

Remarks

This is the same as clicking About in the Properties window.

Example

VB

```
GridControl1.AboutBox
```

C++

```
m_wndGrid.AboutBox();
```

Drag Method (Visual Basic Only)

Begins, ends, or cancels a drag operation for the InaGrid control. Doesn't support named arguments.

Syntax

*Object.***Drag** action

The Drag method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
action (optional)	(Optional) A constant or value that specifies the action to perform, as described in Settings. If action is omitted, the default is to begin dragging the object.

Settings

The settings for action are:

Constant	Value	Description
vbCancel	0	Cancels drag operation
vbBeginDrag	1	Begins dragging object
vbEndDrag	2	Ends dragging and drop object

Remarks

These constants are listed in the Visual Basic (VB) object library in the Object Browser .

Using the Drag method to control a drag-and-drop operation is required only when the DragMode property of the object is set to Manual (0). However, you can use Drag on an object whose DragMode property is set to Automatic (1 or vbAutomatic).

If you want the mouse pointer to change shape while the object is being dragged, use either the DragIcon or MousePointer property. The MousePointer property is only used if no DragIcon is specified.

In earlier versions of Visual Basic, Drag was an asynchronous method where subsequent statements were invoked even though the Drag action wasn't finished.

ForceVisible Method

Makes the specified row visible within the InaGrid control by scrolling the client area.

Syntax

VB

*Object.***ForceVisible** nRow, bPartialOk

C++

void *Object.***ForceVisible**(const CURRENCY& nRow, BOOL bPartialOk)

The **ForceVisible** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
nRow	The row number to make visible.
bPartialOk	A Boolean expression specifying whether the full row must be visible.

Settings

The settings for bPartialOk are:

Setting	Description
True	If any part of the row is visible, scrolling is unnecessary.
False	Require that the InaGrid control scroll so that the entire row is visible.

Remarks

nRow is a 64-bit parameter that is expressed in the CURRENCY data type. You must scale this value by dividing by 10,000.

Example

VB

```
Dim nRow As CURRENCY
nRow = 45
GridControl1.ForceVisible nRow, False
```

C++

```
CURRENCY nRow;
nRow.int64 = 45;
m_wndGrid.ForceVisible(nRow, FALSE);
```

ForceVisibleColumn Method

Makes the specified column visible within the InaGrid control by scrolling the client area.

Syntax

VB

*Object.***ForceVisibleColumn** pColumnHeader, bPartialOk

C++

void *Object.***ForceVisibleColumn**(LPDISPATCH pColumnHeader, BOOL bPartialOk)

The **ForceVisibleColumn** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
pColumnHeader	The column to make visible.
bPartialOk	A Boolean expression specifying whether the full column must be visible.

Settings

The settings for bPartialOk are:

Setting	Description
True	If any part of the row is visible, scrolling is unnecessary.
False	Require that the InaGrid control scroll so that the entire row is visible.

Example

VB

```
'Force the second column to be fully visible
Dim aColHdrs As ColumnHeaders
Set aColHdrs = GridControl1.ColumnHeaders
GridControl1.ForceVisibleColumn aColHdrs.Item(1), False
```

C++

```
// Force the second column to be fully visible
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
CColumnHeader header = hdrCol.GetItem(1);
m_wndGrid.ForceVisibleColumn(header, FALSE);
```

GetColumnNumber Method

Returns the column number that the specified horizontal pixel resides in.

Syntax

VB

*object.***GetColumnNumber** xPos

C++

long *Object.***GetColumnNumber**(long xPos)

The **GetColumnNumber** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
xPos	The horizontal pixel position relative to the InaGrid client area.

Remarks

Returns -1 if the specified pixel does not reside in a column within the InaGrid control.

Example

VB

```
Dim xPos As Integer, lCol As Long
xPos = 50
lCol = GridControl1.GetColumnNumber(xPos)
```

C++

```
long lCol = m_wndGrid.GetColumnNumber(50);
```

GetPaintPageCount Method

Returns the number of pages that would be printed using the current print rectangle.

Syntax

VB

*Object.***GetPaintPageCount** hDc, hDeviceDc, bScaleToFit

C++

CURRENCY *Object.***GetPaintPageCount**(long hDc, long hDeviceDc, BOOL bScaleToFit)

The **GetPaintPageCount** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
hDc	Handle to the printer device context.
hDeviceDc	Handle to the attribute query device context.
bScaleToFit	A Boolean expression that specifies whether the entire print region should be printed on a single page.

Settings

The settings for bScaleToFit are:

Setting	Description
True	Scale the print rectangle to fit on a single page.
False	The print rectangle is printed without scaling and may

occupy multiple pages.

Remarks

If no print rectangle has been set using SetPaintRect, the entire InaGrid control area is considered the print rectangle.

Example

C++

```
CURRENCY PageCount = m_wndGrid.GetPrintPageCount((long)pDC->GetSafeHdc(),
    (long)pDC->m_hAttribDC, FALSE);
```

GetPaintRect Method

Gets the values that define the printing area for subsequent printing using the Print method.

Syntax

VB

*Object.***GetPaintRect** xPos, yPos, xSize, ySize

C++

void *Object.***GetPaintRect**(long* xPos, long* yPos, long* xSize, long* ySize)

The **GetPaintRect** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
xPos	Receives the horizontal pixel origin relative to the InaGrid client area, that defines the left-most printing rectangle.
yPos	Receives the vertical pixel origin relative to the InaGrid client area, that defines the top-most printing rectangle.
xSize	Receives the horizontal pixel width relative to the origin, that defines the width of the printing rectangle.
ySize	Receives the vertical pixel height relative to the origin, that defines the height of the printing rectangle.

Example

VB

```
Dim xPos As Long, yPos As Long, xSize As Long, ySize As Long
GridControl1.GetPaintRect xPos, yPos, xSize, ySize
```

C++

```
long xPos, yPos, xSize, ySize;
m_wndGrid.GetPaintRect(&xPos, &yPos, &xSize, &ySize);
```

GetPaintWidthCount Method

Returns the number of pages that would be required to print the width of the current print rectangle.

Syntax

VB

*Object.***GetPaintWidthCount** hDc, hDeviceDc

C++

long *Object.***GetPaintWidthCount**(long hDc, long hDeviceDc)

The **GetPaintWidthCount** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
hDc	Handle to the printer device context.
hDeviceDc	Handle to the attribute query device context.

Remarks

If no print rectangle has been set using SetPaintRect, the entire InaGrid control area is considered the print rectangle.

Example

C++

```
long cxWidthCount = m_wndGrid.GetPaintWidthCount((long)pDC->GetSafeHdc(),
    (long)pDC->m_hAttribDC);
```

GetRowNumber Method

Returns the row number that the specified vertical pixel resides in.

Syntax

VB

*Object.***GetRowNumber** yPos

C++

CURRENCY *Object.***GetRowNumber**(long yPos)

The **GetRowNumber** method syntax has these parts:

Part	Description

Object	An object expression that evaluates to an InaGrid control.
yPos	The vertical pixel position relative to the InaGrid client area.

Remarks

Returns a 64-bit parameter that is expressed in the CURRENCY data type. You must scale this value by dividing by 10,000.

Returns -1 if the specified pixel does not reside in a row within the InaGrid control.

Example

VB

```
Dim yPos As Integer, nRow As CURRENCY
yPos = 123
nRow = GridControl1.GetRowNumber(yPos)
```

C++

```
CURRENCY nRow = m_wndGrid.GetRowNumber(123);
```

Move Method (Visual Basic Only)

Moves an InaGrid control. Doesn't support named arguments.

Syntax

object.**Move** left, top, width, height

The **Move** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
left	(Required) Single-precision value indicating the horizontal coordinate (x-axis) for the left edge of the InaGrid control.
top	(Optional) Single-precision value indicating the vertical coordinate (y-axis) for the top edge of the InaGrid control.
width	(Optional) Single-precision value indicating the new width of the InaGrid control.
height	(Optional) Single-precision value indicating the new height of the InaGrid control.

Remarks

Only the left argument is required. However, to specify any other arguments, you must specify all arguments that appear in the syntax before the argument you want to specify. For example, you can't specify width without specifying left and top. Any trailing arguments that are unspecified remain unchanged.

When the InaGrid control is in a Frame control, the coordinate system is always in twips. Moving the control in a Frame is always relative to the origin (0,0), which is the upper-left corner. When moving an InaGrid control on a Form object or in a PictureBox (or an MDI child form on an MDIForm object), the coordinate system of the container object is used. The coordinate system or unit of measure is set with the ScaleMode property at design time. You can change the coordinate system at run time with the Scale method.

Paint Method

Prints a single page within the current print rectangle of the InaGrid control.

Syntax

VB

*Object.***Paint** hDc, hDeviceDc, nRow, nHeader, bScaleToFit

C++

void *Object.***Paint**(long hDc, long hDeviceDc, CURRENCY* pnRow, long* pnHeader, BOOL bScaleToFit)

The **Paint** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
hDc	Handle to the printer device context.
hDeviceDc	Handle to the attribute query device context.
nRow	Passes in the top-most row number to begin printing and receives the last row number that was printed on the page.
nHeader	Passes in the left-most column number to begin printing and receives the last column number that was printed on the page.
bScaleToFit	A Boolean expression that specifies whether the entire print region should be scaled to fit on a single page.

Settings

The settings for bScaleToFit are:

Setting	Description
True	Scale the print rectangle to fit on a single page.
False	The print rectangle is printed without scaling and may occupy multiple pages.

Remarks

nRow is a 64-bit parameter that is expressed in the CURRENCY data type. You must scale this value by multiplying by 10,000.

If no print rectangle has been set using SetPaintRect, the entire InaGrid control area is printed.

Example

C++

```
CURRENCY Row;
long nHeader = 0;

Row.int64 = 0;
m_wndGrid.Print((long)pDC->GetSafeHdc(), (long)pDC->m_hAttribDC, &Row,
&nHeader, FALSE);
```

PrintIt Method

Prints as many pages as required to output all the data to a printer.

Syntax

VB

*Object.***PrintIt** hDeviceDC As OLE_HANDLE, nStartRow As CURRENCY, nEndRow As CURRENCY, bScaleToFit As Boolean

VC

void *Object.***PrintIt**(OLE_HANDLE hDeviceDc, CURRENCY nStartRow, CURRENCY nEndRow, VARIANT_BOOL bScaleToFit);

The **PrintIt** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
hDeviceDc	Handle to the printer device context.
nStartRow	Passes in the top-most row number to begin printing.
nEndRow	Passes in the bottom-most row number to end printing.
bScaleToFit	A Boolean expression that specifies whether the entire print region should be printed on a single page.

Settings

The settings for bScaleToFit are:

Setting	Description
True	Scale the columns to fit on a single page.
False	The print rectangle is printed without scaling and may occupy multiple pages.

Example

VB

```
Private Sub mnuFilePrint_Click()
Dim Grid As INAGRIDLib.GridControl
Dim NumCopies, i
```

```
‘ Get the active inagrid control from a MDI child frame
Set Grid = ActiveForm.ActiveControl
‘ Set Cancel to True
dlgCommonDialog.CancelError = True
On Error GoTo ErrHandler
‘ Display the Print dialog box
dlgCommonDialog.ShowPrinter
‘ Get user-selected values from the dialog box
NumCopies = dlgCommonDialog.Copies
For i = 1 To NumCopies
‘ Just Print It
Grid.PrintIt Printer.hdc, 0, Grid.Count, False
Next i
Exit Sub
ErrHandler:
    ‘ User pressed the Cancel button
    Exit Sub
End Sub
```

C++

```
CPrintInfo printInfo;
if (NULL == printInfo.m_pPD->m_pd.hDC)
{
    AfxGetApp()->GetPrinterDeviceDefaults(&printInfo.m_pPD->m_pd);
       if ((NULL == printInfo.m_pPD->m_pd.hDC) && (NULL ==
printInfo.m_pPD->CreatePrinterDC()))
    return;
}
CY nStartRow;
nStartRow.int64 = 0;
CY nEndRow = m_wndGrid.GetCount();
nEndRow.int64 -= 1;
m_wndGrid.PrintIt((OLE_HANDLE)printInfo.m_pPD->m_pd.hDC, nStartRow, nEndRow,
TRUE);
```

SetFocus Method (Visual Basic Only)

Moves the focus to the InaGrid control.

Syntax

object.**SetFocus**

The object placeholder is an object expression that evaluates to an InaGrid control.

Remarks

After invoking the SetFocus method, any user input is directed to the InaGrid control.

You can only move the focus to an InaGrid control that is visible.

You also can't move the focus to an InaGrid control if the Enabled property is set to False. If the Enabled property has been set to False at design time, you must first set it to True before it can receive the focus using the SetFocus method.

SetPaintRect Method

Sets the rectangle that defines the printing area for subsequent printing using the Paint method.

Syntax

VB

*Object.***SetPaintRect** xPos, yPos, xSize, ySize

C++

void *Object.***SetPaintRect**(long xPos, long yPos, long xSize, long ySize)

The **SetPrintRect** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
xPos	The horizontal pixel origin relative to the InaGrid client area, that defines the left-most printing rectangle.
yPos	The vertical pixel origin relative to the InaGrid client area, that defines the top-most printing rectangle.
xSize	The horizontal pixel width relative to the origin, that defines the width of the printing rectangle.
ySize	The vertical pixel height relative to the origin, that defines the height of the printing rectangle.

Example

VB

```
GridControl1.SetPaintRect 0, 0, 400, 200
```

C++

```
int nPageWidth = pDC->GetDeviceCaps(HORZRES);
int nPageHeight     = pDC->GetDeviceCaps(VERTRES);
CSize  szInch(pDC->GetDeviceCaps(LOGPIXELSX), pDC->GetDeviceCaps(LOGPIXELSY));
CRect  rect(0, 0, nPageWidth, nPageHeight);

rect.InflateRect(-szInch.cx/2, -szInch.cy/2);
m_wndGrid.SetPaintRect(rect.left, rect.top, rect.Width(), rect.Height());
```

ShowWhatsThis Method (Visual Basic Only)

Displays a selected topic in a Help file using the What's This popup provided by Windows 95 Help.

Syntax

object.**ShowWhatsThis**

The object placeholder is an object expression that evaluates to an InaGrid control.

Remarks

The ShowWhatsThis method is very useful for providing context-sensitive Help from a context menu in your application. The method displays the topic identified by the WhatsThisHelpID property of the object specified in the syntax.

UpdateColumn Method

Forces a complete repaint of a column in an InaGrid control.

Syntax

VB

*Object.***UpdateColumn** nColumn

C++

void *Object.***UpdateColumn**(long nColumn)

The **UpdateColumn** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
nColumn	A numeric expression specifying the column number.

Remarks

The value of nColumn can range from 0 to object.ColumnHeaders.Count -1.

UpdateRow Method

Forces a complete repaint of a row in an InaGrid control.

Syntax

VB

*Object.***UpdateRow** nRow

C++

void *Object.***UpdateRow**(const CURRENCY& nRow)

The **UpdateRow** method syntax has these parts:

Part	Description

Object	An object expression that evaluates to an InaGrid control.
nRow	A numeric CURRENCY expression specifying the row number.

Remarks

This method is used to force the control to redisplay data for the specified row. The data is refreshed during the GetData event. The value of nRow can range from 0 to Count -1.

nRow is a 64-bit parameter that is expressed in the CURRENCY data type. You must scale this value by dividing by 10,000.

Example

VB

```
InaGridControl.UpdateRow 20 / 10000
```

C++

```
CURRENCY Row;
Row.int64 = 20;
m_wndGrid.UpdateRow(Row);
```

ZOrder Method (Visual Basic Only)

Places an InaGrid control at the front or back of the z-order within its graphical level. Doesn't support named arguments.

Syntax

*Object.***ZOrder** position

The ZOrder method syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
Position	(Optional) Integer indicating the position of the control relative to other instances of the same control. If position is 0 or omitted, object is positioned at the front of the z-order. If position is 1, object is positioned at the back of the z-order.

Remarks

The z-order of InaGrid controls can be set at design time by choosing the Bring To Front or Send To Back menu command from the Edit menu.

Three graphical layers are associated with forms and containers. The back layer is the drawing space where the results of the graphics methods are displayed. Next is the middle layer where graphical objects and Label controls are displayed. The front layer is where all nongraphical controls like the InaGrid control are displayed. Anything contained in a layer closer to the front covers anything contained in the layer(s) behind it. ZOrder arranges objects only within the layer where the object is displayed.

Events

Click Event

Occurs when the user presses and then releases a mouse button over an InaGrid control. It can also occur when the value of the control is changed.

For an InaGrid control, this event occurs when the user clicks the control with the left or right mouse button.

Syntax

VB

Private Sub object**_Click**([index As Integer])

C++

void **OnClick**()

The Click event syntax has these parts:

Part	Description
object	An object expression that evaluates to an InaGrid control.
index	An integer that uniquely identifies the control if it's in a control array.

Remarks

Note To distinguish between the left, right, and middle mouse buttons, use the MouseDown and MouseUp events.

If there is code in the Click event, the DblClick event will never trigger, because the Click event is the first event to trigger between the two. As a result, the mouse click is intercepted by the Click event, so the DblClick event doesn't occur.

DblClick Event

Occurs when the user presses and releases a mouse button and then presses and releases it again over an InaGrid control.

Syntax

VB

Private Sub object_DblClick (index As Integer)

C++

void **OnClick**()

Part	Description
object	An object expression that evaluates to an InaGrid control.
index	Identifies the control if it's in a control array .

Remarks

The argument *Index* uniquely identifies the control if it's in a control array. You can use a DblClick event procedure for an implied action or to carry out multiple steps with a single action

The mouse events occur in this order: MouseDown, MouseUp, Click, DblClick, and MouseUp.

If DblClick doesn't occur within the system's double-click time limit, the object recognizes another Click event. The double-click time limit may vary because the user can set the double-click speed in the Control Panel. When you're attaching procedures for these related events, be sure that their actions don't conflict.

Note To distinguish between the left, right, and middle mouse buttons, use the MouseDown and MouseUp events.

If there is code in the Click event, the DblClick event will never trigger.

DragDrop Event (Visual Basic Only)

See Also FAQ Properties Methods Events Enums

Occurs when a drag-and-drop operation is completed as a result of dragging a control over an object and releasing the mouse button or using the Drag method with its action argument set to 2 (Drop).

Syntax

Private Sub **object_DragDrop**([index As Integer,]source As Control, x As Single, y As Single)

The DragDrop event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
index	An integer that uniquely identifies a control if it's in a control array.
source	The control being dragged. You can include properties and methods in the event procedure with this argument —for example, Source.Visible = 0.
x, y	A number that specifies the current horizontal (x) and

	vertical (y) position of the mouse pointer within the target form or control. These coordinates are always expressed in terms of the target's coordinate system as set by the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties.

Remarks

Use a DragDrop event procedure to control what happens after a drag operation is completed. For example, you can move the source control to a new location or copy a file from one location to another.

When multiple controls can potentially be used in a source argument:

- Use the TypeOf keyword with the If statement to determine the type of control used with source.
- Use the control's Tag property to identify a control, and then use a DragDrop event procedure.

Note Use the DragMode property and Drag method to specify the way dragging is initiated. Once dragging has been initiated, you can handle events that precede a DragDrop event with a DragOver event procedure.

DragOver Event (Visual Basic Only)

Occurs when a drag-and-drop operation is in progress. You can use this event to monitor the mouse pointer as it enters, leaves, or rests directly over a valid target. The mouse pointer position determines the target object that receives this event.

Syntax

Private Sub **object_DragOver**([index As Integer,]source As Control, x As Single, y As Single, state As Integer)

The DragOver event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
index	An integer that uniquely identifies a control if it's in a control array.
source	The control being dragged. You can refer to properties and methods in the event procedure with this argument--for example, Source.Visible = False.
x, y	A number that specifies the current horizontal (x) and vertical (y) position of the mouse pointer within the target form or control. These coordinates are always expressed in terms of the target's coordinate system as set by the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties.
state	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control:

0 = Enter (source control is being dragged within the range of a target).

1 = Leave (source control is being dragged out of the range of a target).

2 = Over (source control has moved from one position in the target to another).

Remarks

Use a DragOver event procedure to determine what happens after dragging is initiated and before a control drops onto a target. For example, you can verify a valid target range by highlighting the target (set the BackColor or ForeColor property from code) or by displaying a special drag pointer (set the DragIcon or MousePointer property from code).

Use the state argument to determine actions at key transition points. For example, you might highlight a possible target when state is set to 0 (Enter) and restore the object's previous appearance when state is set to 1 (Leave).

When an object receives a DragOver event while state is set to 0 (Enter):

- If the source control is dropped on the object, that object receives a DragDrop event.
- If the source control isn't dropped on the object, that object receives another DragOver event when state is set to 1 (Leave).

Note Use the DragMode property and Drag method to specify the way dragging is initiated. For suggested techniques with the source argument, see Remarks for the DragDrop event topic.

GetData Event

Occurs when the InaGrid control is about to display row and column data.

Syntax

VB

Private Sub **object_GetData**(nRow As CURRENCY, pColumn As ColumnHeader, pValue As String, pColor As OLE_COLOR)

C++

void **OnGetData**(CURRENCY nRow, LPDISPATCH pColumn, BSTR FAR* pValue, OLE_COLOR* pColor);

The GetData event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
nRow	The row that requires data.
pColumn	The column that requires data.

pValue	Specifies the data for the row and column.
pColor	Specifies the color of the text used to display the data.

Remarks

This event occurs when a cell in the InaGrid control is about to display its data. Use this event to populate the InaGrid control with the data from your application. Because the control does not keep a separate copy of the application's data, it must callback into the application, using this event, to display its data. This allows a very large number of rows and columns to be displayed in a virtual view. This event is only called to display the rows and columns that are required by the current position of the control.

Example

VB

```
Private Sub GridEvts_GetData(ByVal nRow As INAGRIDLib.CURRENCY, ByVal pColumn
As INAGRIDLib.IColumnHeader, pValue As String, pColor As Stdole.OLE_COLOR)

    'Simply put the row and column number as data to the grid
    If pColumn Is Nothing Then
        pValue = Str((nRow * 10000) + 1)
    Else
        pValue = "Row = " + Str(nRow * 10000) + ", Col = " + Str(pColumn.Id)
    End If
End Sub
```

C++

```
void CInaGridContainer::OnGetData(CURRENCY nRow, LPDISPATCH pColumn, BSTR FAR*
pValue, OLE_COLOR* pColor)
{
    CString strData;
    if(NULL == pColumn)
    {
        strData.Format("%d", (int)nRow.int64 + 1);
        *pColor = (0x80000000 | COLOR_BTNTEXT);
    }
    else
    {
        CColumnHeader hdrColumn(pColumn);
        strData.Format("Row=%d, Col = %d", (int)nRow.int64,
hdrColumn.GetId());
        hdrColumn.DetachDispatch();
        *pColor = (0x80000000 | COLOR_WINDOWTEXT);
    }
    strData.SetSysString(pValue);
}
```

GetFormat Event

Occurs when the InaGrid control is about to display row and column data.

Syntax

VB

Private Sub *object*_**GetFormat**(nRow As CURRENCY, pColumn As ColumnHeader, pFormat As FormatData)

C++

void *object*::**OnGetFormat**(CURRENCY nRow, LPDISPATCH pColumn, LPDISPATCH pFormat);

The GetData event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
nRow	The row that requires data.
pColumn	The column that requires data.
pFormat	Specifies the data for the row and column.

Remarks

This event occurs when a cell in the InaGrid control is about to display its data. Use this event to populate the InaGrid control with the data from your application. Because the control does not keep a separate copy of the application's data, it must callback into the application, using this event, to display its data. This allows a very large number of rows and columns to be displayed in a virtual view. This event is only called to display the rows and columns that are required by the current position of the control.

Example

VB

```
Private Sub GridEvts_GetFormat(ByVal nRow As INAGRIDLib.CURRENCY, ByVal
pColumn As INAGRIDLib.IColumnHeader, ByVal pFormat As INAGRIDLib.IFormatData)

pFormatData.HorizontalAlignment = eHAlign;

End Sub
```

C++

```
void OnGetFormat(CURRENCY nRow, LPDISPATCH pColumn, LPDISPATCH pFormatData)

{
       CFormatData formatData(pFormatData);
       formatData.SetHorizontalAlignment(cellInfo.m_eHAlign);
       formatData.SetVerticalAlignment(cellInfo.m_eVAlign);
       formatData.SetWrapText(cellInfo.m_bWrapText);
       formatData.SetTextColor(cellInfo.m_dwTextColor);
       formatData.SetBackColor(cellInfo.m_dwBackColor);
       formatData.SetFontName(cellInfo.m_strFontName);
       formatData.SetFontSize((short)cellInfo.m_nFontSize);
       formatData.SetFontBold(cellInfo.m_bFontBold);
       formatData.SetFontItalic(cellInfo.m_bFontItalic);
       formatData.SetFontUnderline(cellInfo.m_bFontUnderline);
       formatData.SetFontStrikethrough(cellInfo.m_bFontStrikeout);
       formatData.SetFontCharSet(cellInfo.m_nFontCharSet);
       formatData.DetachDispatch();
}
```

GotFocus Event (Visual Basic Only)

Occurs when an InaGrid control receives the focus, either by user action, such as tabbing to or clicking the control, or by changing the focus in code using the SetFocus method.

Syntax

Private Sub **object_GotFocus**([index As Integer])

The GotFocus event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
index	An integer that uniquely identifies a control if it's in a control array.

Remarks

Typically, you use a GotFocus event procedure to specify the actions that occur when the InaGrid control first receives the focus. For example, by attaching a GotFocus event procedure to an InaGrid control on a form, you can guide the user by displaying brief instructions or status bar messages. You can also provide visual cues by enabling, disabling, or showing other controls that depend on the InaGrid control that has the focus.

Note An InaGrid control can receive the focus only if its Enabled and Visible properties are set to True. To customize the keyboard interface in Visual Basic for moving the focus, set the tab order or specify access keys for the control.

IsSelected Event

Occurs when a row or cell is displayed.

Syntax

VB

Private Sub **object_IsSelected**(nRow As CURRENCY, pColumn As ColumnHeader, bSelect As Boolean)

C++

void **OnIsSelected**(CURRENCY nRow, LPDISPATCH pColumn, BOOL FAR* bSelect);

The IsSelected event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.

nRow	The row that is being displayed.
pColumn	The column that is being displayed.
bSelect	Specifies whether the row or cell should be selected.

Remarks

This event occurs when a cell in the InaGrid control is displayed. Using data passed in the parameters to this event, the row and column information can be determined and the selection can be made or cancelled if desired.

Example

VB

```
Private Sub InaGridControl_IsSelected()(ByVal nRow As INAGRIDLib.CURRENCY,
ByVal pColumn As INAGRIDLib.IColumnHeader, pValue As Boolean)
pValue = False
If Not pColumn Is Nothing Then
If (nRow = (SelectedRow / 10000)) And (pColumn.Id = SelectedColumn) Then
pValue = True
End If
End Sub
```

C++

```
void CInaGridContainer::OnIsSelected(CURRENCY nRow, LPDISPATCH pColumn, BOOL
FAR* pValue)
{
    *pValue = FALSE;
    if(NULL != pColumn)
    {
        CColumnHeader hdrColumn(pColumn);
        *pValue = ((nRow.int64 == m_n64SelectedRow) && (hdrColumn.GetId() ==
m_nSelectedColumn));
        hdrColumn.DetachDispatch();
    }
}
```

KeyDown, KeyUp Events

Occur when the user presses (KeyDown) or releases (KeyUp) a key while an InaGrid control has the focus. (To interpret ANSI characters, use the KeyPress event.)

Syntax

VB

Private Sub *object*_**KeyDown(**[*index* **As Integer**,]*keycode* **As Integer**, *shift* **As Integer)**

Private Sub *object*_**KeyUp(**[*index* **As Integer**,]*keycode* **As Integer**, *shift* **As Integer)**

C++

void **OnKeyDown**(short* keycode, short shift)

void **OnKeyUp**(short* keycode, short shift)

The KeyDown and KeyUp event syntaxes have these parts:

Part	Description
object	An object expression that evaluates to an InaGrid control.
index	An integer that uniquely identifies a control if it's in a control array.
keycode	A key code, such as **vbKeyF1** (the F1 key) or **vbKeyHome** (the HOME key). To specify key codes, use the constants in the Visual Basic (VB) object library in the Object Browser.
shift	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Remarks

For both events, the InaGrid control with the focus receives all keystrokes. Although the KeyDown and KeyUp events can apply to most keys, they're most often used for:

- Extended character keys such as Function Keys .
- Navigation keys.
- Combinations of keys with standard keyboard modifiers.
- Distinguishing between the numeric keypad and regular number keys.

Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key.

KeyDown and KeyUp are not invoked for the TAB key.

KeyDown and KeyUp interpret the uppercase and lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key) and *shift*, which indicates the state of *shift+key* and therefore returns either A or a.

If you need to test for the *shift* argument, you can use the *shift* constants, which define the bits within the argument. The constants have the following values:

Constant	Value	Description
vbShiftMask	1	SHIFT key bit mask.
VbCtrlMask	2	CTRL key bit mask.
VbAltMask	4	ALT key bit mask.

The constants act as bit masks that you can use to test for any combination of keys.

You test for a condition by first assigning each result to a temporary integer variable and then comparing *shift* to a bit mask. Use the **And** operator with the *shift* argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

```
ShiftDown = (Shift And vbShiftMask) > 0
```

In a procedure, you can test for any combination of conditions, as in this example:

```
If ShiftDown And CtrlDown Then
```

KeyPress Event

Occurs when the user presses and releases an ANSI key.

Syntax

VB

Private Sub *object*_**KeyPress(**[*index* **As Integer**,]*keyascii* **As Integer)**

C++

void **OnKeyPress**(short* keyascii)

The KeyPress event syntax has these parts:

Part	Description
object	An object expression that evaluates to an InaGrid control.
index	An integer that uniquely identifies the control if it's in a control array .
keyascii	An integer that returns a standard numeric ANSI keycode. keyascii is passed by reference; changing it sends a different character to the object. Changing keyascii to 0 cancels the keystroke so the object receives no character.

Remarks

The InaGrid control with the focus receives the event. A KeyPress event can involve any printable keyboard character, the CTRL key combined with a character from the standard alphabet or one of a few special characters, and the ENTER or BACKSPACE key.

You can convert the *keyascii* argument into a character by using the expression:

```
Chr(KeyAscii)
```

You can then perform string operations and translate the character back to an ANSI number that the control can interpret by using the expression:

```
KeyAscii = Asc(char)
```

Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress doesn't indicate the physical state of the keyboard; instead, it passes a character.

KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters. KeyDown and KeyUp interpret the uppercase and

lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key), and *shift*, which indicates the state of *shift+key* and therefore returns either A or a.

If the **KeyPreview** property is set to **True**, a form receives the event before controls on the form receive the event. Use the **KeyPreview** property to create global keyboard-handling routines.

Note The ANSI number for the keyboard combination of CTRL+@ is 0. Because Visual Basic recognizes a *keyascii* value of 0 as a zero-length string (""), avoid using CTRL+@ in your applications.

LostFocus Event (Visual Basic Only)

Occurs when an InaGrid control loses the focus, either by user action, such as tabbing to or clicking another object, or by changing the focus in code using the SetFocus method.

Syntax

Private Sub **object_LostFocus**([index As Integer])

The LostFocus event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
index	An integer that uniquely identifies a control if it's in a control array.

Remarks

A LostFocus event procedure is primarily useful for verification and validation updates. Using LostFocus can cause validation to take place as the user moves the focus from the control. Another use for this type of event procedure is enabling, disabling, hiding, and displaying other objects as in a GotFocus event procedure. You can also reverse or change conditions that you set up in the object's GotFocus event procedure.

MouseDown, MouseUp Events

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button.

Syntax

VB

Private Sub object_MouseDown([index As Integer,]button As Integer, shift As Integer, x As Single, y As Single)

Private Sub object _MouseUp([index As Integer,]button As Integer, shift As Integer, x As Single,

y As Single)

C++

void **OnMouseDown**(short button, short shift, long x, long y)

void **OnMouseUp**(short button, short shift, long x, long y)

The MouseDown and MouseUp event syntaxes have these parts:

Part	Description
object	An object expression that evaluates to an InaGrid control.
index	Returns an integer that uniquely identifies the control if it's in a control array.
button	Returns an integer that identifies the button that was pressed (MouseDown) or released (MouseUp) to cause the event. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
shift	Returns an integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released. A bit is set if the key is down. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of shift would be 6.
x, y	Returns a number that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the InaGrid control

Remarks

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a given mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

The following applies to both Click and DblClick events:

- If a mouse button is pressed while the pointer is over an InaGrid control, then it "captures" the mouse and receives all mouse events up to and including the last MouseUp event. This implies that the *x*, *y* mouse-pointer coordinates returned by a mouse event may not always be in the internal area of the object that receives them.
- If mouse buttons are pressed in succession, the control that captures the mouse after the first press receives all mouse events until all buttons are released.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the Visual Basic (VB) object library in the Object Browser_to define the bits within the argument:

Constant (Button)	Value	Description
vbLeftButton	1	Left button is pressed
vbRightButton	2	Right button is pressed
vbMiddleButton	4	Middle button is pressed

Constant (Shift)	Value	Description
vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

Note You can use a MouseMove event procedure to respond to an event caused by moving the mouse. The *button* argument for MouseDown and MouseUp differs from the *button* argument used for MouseMove. For MouseDown and MouseUp, the *button* argument indicates exactly one button per event, whereas for MouseMove, it indicates the current state of all buttons.

MouseMove Event

Occurs when the user moves the mouse.

Syntax

VB

Private Sub object_**MouseMove**([index As Integer,] button As Integer, shift As Integer, x As Single, y As Single)

C++

void **OnMouseMove**(short button, short shift, long x, long y)

The MouseMove event syntax has these parts:

Part	Description
object	An object expression that evaluates to an InaGrid control.
index	An integer that uniquely identifies the control if it's in a control array .

button	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. It indicates the complete state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are pressed.
shift	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys. A bit is set if the key is down. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of shift would be 6.
x, y	A number that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Remarks

The MouseMove event is generated continually as the mouse pointer moves across the InaGrid control. Unless another object has captured the mouse, the control recognizes a MouseMove event whenever the mouse position is within its borders.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the Visual Basic (VB) object library in the Object Browser to define the bits within the argument:

Constant (Button)	Value	Description
vbLeftButton	1	Left button is pressed.
vbRightButton	2	Right button is pressed.
vbMiddleButton	4	Middle button is pressed.

Constant (Shift)	Value	Description
vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

You test for a condition by first assigning each result to a temporary integer variable and then comparing the *button* or *shift* arguments to a bit mask. Use the **And** operator with each argument to test if the condition is greater than zero, indicating the key or button is pressed, as in this example:

```
LeftDown = (Button And vbLeftButton) > 0
```

```
CtrlDown = (Shift And vbCtrlMask) > 0
```

Then, in a procedure, you can test for any combination of conditions, as in this example:

```
If LeftDown And CtrlDown Then
```

Note You can use MouseDown and MouseUp event procedures to respond to events caused by pressing and releasing mouse buttons.

The *button* argument for MouseMove differs from the *button* argument for MouseDown and MouseUp. For MouseMove, the *button* argument indicates the current state of all buttons; a single MouseMove event can indicate that some, all, or no buttons are pressed. For MouseDown and MouseUp, the *button* argument indicates exactly one button per event.

Any time you move a window inside a MouseMove event, it can cause a cascading event . MouseMove events are generated when the window moves underneath the pointer. A MouseMove event can be generated even if the mouse is perfectly stationary.

OLECompleteDrag Event

Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled.

Syntax

VB

Private Sub *object*_**OLECompleteDrag**(pEffect As Long)

C++

void *object*::**OnOLECompleteDrag** (long FAR* pEffect)

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Effect	A long integer set by the source object identifying the action that has been performed, thus allowing the source to take appropriate action if the component was moved (such as the source deleting data if it is moved from one component to another). The possible values are listed in Settings.

Settings

The settings for effect are:

Constant	Value	Description

gOLEDropEffectNone	0	Drop target cannot accept the data, or the drop operation was cancelled.
gOLEDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
gOLEDropEffectMove	2	Drop results in a link to the original data being created between drag source and drop target.

Example

VB

```
Private Sub TopGrid_OLECompleteDrag(pEffect As Long)
    TopDragDropTarget.EndDrag pEffect
End Sub
```

C++

```
void CDragDropGrid::EndDrag(long* pdwEffect)
{
       if ((*pdwEffect & DROPEFFECT_MOVE) != 0)
       {
          m_Values.erase(m_Values.begin() + m_DragSelection);
          Resize();
       }
}
```

Remarks

The OLECompleteDrag event is the final event to be called in an OLE drag/drop operation. This event informs the source component of the action that was performed when the object was dropped onto the target component. The target sets this value through the effect parameter of the OLEDragDrop event. Based on this, the source can then determine the appropriate action it needs to take. For example, if the object was moved into the target (vbDropEffectMove), the source needs to delete the object from itself after the move.

If OLEDragMode is set to Automatic, then Visual Basic handles the default behavior. The event still occurs, however, allowing the user to add to or change the behavior.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

OLEDragDrop Event

Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.

Note This event occurs only if OLEDropMode is set to 1 (Manual).

Syntax

VB

Private Sub *object*_***OLEDragDrop***(Data As INAGRIDLib.DataObject, pEffect As Long, ByVal nButton As Integer, ByVal nShift As Integer, ByVal x As Single, ByVal y As Single)

C++

void *object*::**OnOLEDragDrop** (DataObject** ppData, long FAR* pEffect, short nButton, short nShift, long x, long y)

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
data	A DataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the DataObject, it is provided when the control calls the GetData method. The SetData and Clear methods cannot be used here.
effect	A long integer set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed in Settings.
button	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed.
shift	An integer which acts as a bit field corresponding to the state of the SHIFT, CTRL, and ALT keys when they are depressed. The SHIFT key is bit 0, the CTRL key is bit 1, and the ALT key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys were depressed, the value of shift would be 6.
x,y	A number which specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties of the object.

Settings

The settings for effect are:

Constant	**Value**	**Description**
gOLEDropEffectNone	0	Drop target cannot accept the data.
gOLEDropEffectCopy	1	Drop results in a copy of data from

		the source to the target. The original data is unaltered by the drag operation.
gOLEDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

Example

VB

```
Private Sub TopGrid_OLEDragDrop(Data As INAGRIDLib.DataObject, pEffect As
Long, ByVal nButton As Integer, ByVal nShift As Integer, ByVal x As Single,
ByVal y As Single)
    TopDragDropTarget.Drop Data, pEffect, nButton, nShift, y / VerticalScale
End Sub
```

C++

```
void CDragDropGrid::Drop(IVBDataObject* pData, short nButton, short nShift,
CPoint point, long* pdwEffect)
{
int       iFileCount;
       int    iFileIndex;
       int    iDropIndex;
       _variant_t    vResult;
       BSTR          bsData = 0;
       *pdwEffect &= CanProcess(pData, nButton, nShift);
       if (*pdwEffect != DROPEFFECT_NONE)
       {
          iDropIndex = m_pGrid->GetRowNumber(point.y).int64;
          if (iDropIndex < 0)
             iDropIndex = 0;
          if (iDropIndex >= m_pGrid->Count.int64)
             iDropIndex = -1;
          if (pData->GetFormat(m_TransferFormat))
          {
             vResult = pData->GetData(m_TransferFormat);
             if (vResult.vt == (VT_ARRAY | VT_UI1) &&
BstrFromVector(vResult.parray, &bsData) == S_OK)
             {
                    if (iDropIndex < 0)
                           m_Values.push_back(bsData);
                    else
                           m_Values.insert(m_Values.begin() + iDropIndex++,
bsData);
                    if (m_PreviousSelection >= iDropIndex)
                           ++m_PreviousSelection;
                    if (m_DragSelection >= iDropIndex)
                           ++m_DragSelection;
                    SysFreeString(bsData);
             }
          }
          else
          {
             IVBDataObjectFilesPtr    pFiles = pData->Files;
             iFileCount = pFiles->Count;
             for (iFileIndex = 1; iFileIndex <= iFileCount; ++iFileIndex)
             {
                           if (iDropIndex < 0)
                                  m_Values.push_back((LPCTSTR)pFiles-
>Item[iFileIndex]);
```

```
                            else
                                    m_Values.insert(m_Values.begin() +
iDropIndex++, (LPCTSTR)pFiles->Item[iFileIndex]);
                            if (m_PreviousSelection >= iDropIndex)
                                    ++m_PreviousSelection;
                            if (m_DragSelection >= iDropIndex)
                                    ++m_DragSelection;
                }
            }
            Resize();
        }
        Select(m_PreviousSelection);
}
```

Remarks

The source ActiveX component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. Presently, only three of the 32 bits in the effect parameter are used. In future versions of Visual Basic, however, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, vbDropEffectCopy, such as in this manner:

```
If Effect = vbDropEffectCopy...
```

Instead, the source component should mask for the value or values being sought, such as this:

```
If Effect And vbDropEffectCopy = vbDropEffectCopy...
```

-or-

```
If (Effect And vbDropEffectCopy)...
```

This allows for the definition of new drop effects in future versions of Visual Basic while preserving backwards compatibility with your existing code.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

OLEDragOver Event

Occurs when one component is dragged over another.

Syntax

VB

Private Sub *object*_**OLEDragOver**(data As DataObject, effect As Long, button As Integer, shift As Integer, x As Single, y As Single, state As Integer)

C++

void *object*::**OnOLEDragOver** (IVBDataObject** ppData, long FAR* pEffect, short nButton, short nShift, long x, long y, short nState)

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	A DataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the DataObject, it is provided when the control calls the GetData method. The SetData and Clear methods cannot be used here.
effect	A long integer initially set by the source object identifying all effects it supports. This parameter must be correctly set by the target component during this event. The value of effect is determined by logically Or'ing together all active effects (as listed in Settings). The target component should check these effects and other parameters to determine which actions are appropriate for it, and then set this parameter to one of the allowable effects (as specified by the source) to specify which actions will be performed if the user drops the selection on the component. The possible values are listed in Settings.
button	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed.
shift	An integer which acts as a bit field corresponding to the state of the SHIFT, CTRL, and ALT keys when they are depressed. The SHIFT key is bit 0, the CTRL key is bit 1, and the ALT key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys are depressed, the value of shift would be 6.
x,y	A number that specifies the current horizontal (x) and vertical (y) position of the mouse pointer within the target form or control. The x and y values are always expressed in terms of the coordinate system set by the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties of the object.
state	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control. The possible values are listed in Settings.

Settings

The settings for effect are:

Constant	Value	Description

gOLEDropEffectNone 0 Drop target cannot accept the data.

gOLEDropEffectCopy 1 Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.

gOLEDropEffectMove 2 Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

gOLEDropEffectScroll -2147483648(&H80000000) Scrolling is occuring or about to occur in the target component. This value is used in conjunction with the other values. Note Use only if you are performing your own scrolling in the target component.

The settings for state are:

Constant	Value	Description
gEnter	0	Source component is being dragged within the range of a target.
gLeave	1	Source component is being dragged out of the range of a target.
gOver	2	Source component has moved from one position in the target to another.

Example

VB

```
Private Sub TopGrid_OLEDragOver(Data As INAGRIDLib.DataObject, pEffect As
Long, ByVal nButton As Integer, ByVal nShift As Integer, ByVal x As Single,
ByVal y As Single, ByVal nState As Integer)
    pEffect = TopDragDropTarget.Drag(Data, nButton, nShift, y / VerticalScale,
nState)
End Sub
```

C++

```
DROPEFFECT CDragDropGrid::DragOver(IVBDataObject* pData, short nButton, short
nShift, CPoint point, short nState)
{
       DROPEFFECT   processValue = CanProcess(pData, nButton, nShift);
       if (processValue != DROPEFFECT_NONE)
       {
          switch (nState)
          {
          case gEnter:
             m_PreviousSelection = m_Selection.int64;
          case gOver:
             Select(m_pGrid->GetRowNumber(point.y).int64);
             break;
          case gLeave:
             Select(m_PreviousSelection);
             break;
          }
       }
       return processValue;
}
```

Remarks

Note If the state parameter is vbLeave, indicating that the mouse pointer has left the target, then the x and y parameters will contain zeros.

The source component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. Presently, only three of the 32 bits in the effect parameter are used. In future versions of Visual Basic, however, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, vbDropEffectCopy, such as in this manner:

If Effect = vbDropEffectCopy...

Instead, the source component should mask for the value or values being sought, such as this:

If Effect And vbDropEffectCopy = vbDropEffectCopy...

-or-

If (Effect And vbDropEffectCopy)...

This allows for the definition of new drop effects in future versions of Visual Basic while preserving backwards compatibility with your existing code.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

OLEGiveFeedback Event

Occurs after every OLEDragOver event. OLEGiveFeedback allows the source component to provide visual feedback to the user, such as changing the mouse cursor to indicate what will happen if the user drops the object, or provide visual feedback on the selection (in the source component) to indicate what will happen.

Syntax

VB

Private Sub *object*_**OLEGiveFeedback**(effect As Long, defaultcursors As Boolean)

C++

void *object*::**OnOLEGiveFeedback** (long FAR* pAllowedEffects, BOOL* bDefaultcursors)

Part	**Description**
Object	An object expression that evaluates to an object in the Applies To list.

Effect	A long integer set by the target component in the OLEDragOver event specifying the action to be performed if the user drops the selection on it. This allows the source to take the appropriate action (such as giving visual feedback). The possible values are listed in Settings.
Defaultcursors	A boolean value which determines whether Visual Basic uses the default mouse cursor proved by the component, or uses a user-defined mouse cursor. True (default) = use default mouse cursor. False = do not use default cursor. Mouse cursor must be set with the MousePointer property of the Screen

Settings

The settings for effect are:

Constant	Value	Description
gOLEDropEffectNone	0	Drop target cannot accept the data.
gOLEDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
gOLEDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.
gOLEDropEffectScroll	-2147483648 (&H80000000)	Scrolling is occuring or about to occur in the target component. This value is used in conjunction with the other values. Note Use only if you are performing your own scrolling in the target component.

Remarks

If there is no code in the OLEGiveFeedback event, or if the defaultcursors parameter is set to True, then Visual Basic automatically sets the mouse cursor to the default cursor provided by the component.

The source component should always mask values from the effect parameter to ensure compatibility with future implementations of components. Presently, only three of the 32 bits in the effect parameter are used. In future versions of Visual Basic, however, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, vbDropEffectCopy, such as in this manner:

```
If Effect = vbDropEffectCopy...
```

Instead, the source component should mask for the value or values being sought, such as this:

```
If Effect And vbDropEffectCopy = vbDropEffectCopy...
```

-or-

```
If (Effect And vbDropEffectCopy)...
```

This allows for the definition of new drop effects in future versions of Visual Basic while preserving backwards compatibility with your existing code.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

OLESetData Event

Occurs on an source component when a target component performs the GetData method on the source's DataObject object, but the data for the specified format has not yet been loaded.

Syntax

VB

Private Sub *object*_**OLESetData**(data As DataObject, dataformat As Integer)

C++

void *object*::**OnOLESetData**(IVBDataObject** ppData, short* dataformat)

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Data	A DataObject object in which to place the requested data. The component calls the SetData method to load the requested format.
Dataformat	An integer specifying the format of the data that the target component is requesting. The source component uses this value to determine what to load into the DataObject object.

Remarks

In certain cases, you may wish to defer loading data into the DataObject object of a source component to save time, especially if the source component supports many formats. This event allows the source to respond to only one request for a given format of data. When this event is called, the source should check the format parameter to determine what needs to be loaded and then perform the SetData method on the DataObject object to load the data which is then passed back to the target component.

OLEStartDrag Event

Occurs when a component's OLEDrag method is performed, or when a component initiates an OLE drag/drop operation when the OLEDragMode property is set to Automatic.

This event specifies the data formats and drop effects that the source component supports. It can also be used to insert data into the DataObject object.

Syntax

VB

Private Sub object_OLEStartDrag(ppData As INAGRIDLib.DataObject, pAllowedEffects As Long)

C++

void *object*::**OnOLEStartDrag** (DataObject** ppData, long FAR* pAllowedEffects)

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	A DataObject object containing formats that the source will provide and, optionally, the data for those formats. If no data is contained in the DataObject, it is provided when the control calls the GetData method. The programmer should provide the values for this parameter in this event. The SetData and Clear methods cannot be used here.
Allowedeffects	A long integer containing the effects that the source component supports. The possible values are listed in Settings. The programmer should provide the values for this parameter in this event.

Settings

The settings for allowedeffects are:

Constant (Button)	Value	Description
gOLEDropEffectNone	0	Drop target cannot accept the data.
gOLEDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
gOLEDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

Example

VB

```
Private Sub TopGrid_OLEStartDrag(ppData As INAGRIDLib.DataObject,
pAllowedEffects As Long)
    pAllowedEffects = TopDragDropTarget.StartDrag(ppData)
End Sub
```

C++

```
void CDragDropGrid::StartDrag(IVBDataObject* pData, long* pdwEffect)
{
       _bstr_t             pSelection  = m_Values[m_DragSelection];
       _variant_t    vaData;
       if (VectorFromBstr(pSelection, &vaData.parray) == S_OK)
       {
          vaData.vt = VT_ARRAY | VT_UI1;
          pData->SetData( &vaData, COleVariant((short)m_TransferFormat));
       }
       *pdwEffect = DROPEFFECT_COPY|DROPEFFECT_MOVE;
}
```

Remarks

The source component should logically Or together the supported values and places the result in the allowedeffects parameter. The target component can use this value to determine the appropriate action (and what the appropriate user feedback should be).

The StartDrag event also occurs if the component's OLEDragMode property is set to Automatic. This allows you to add formats and data to the DataObject object after the component has done so. You can also override the default behavior of the component by clearing the DataObject object (using the Clear method) and then adding your data and formats.

You may wish to defer putting data into the DataObject object until the target component requests it. This allows the source component to save time by not loading multiple data formats. When the target performs the GetData method on the DataObject, the source's OLESetData event will occur if the requested data is not contained in the DataObject. At this point, the data can be loaded into the DataObject, which will in turn provide the data to the target.

If the user does not load any formats into the DataObject, then the drag/drop operation is canceled.

OnDraw Event

Occurs when an OwnerDraw cell is about to be displayed.

Syntax

VB

Private Sub **object_OnDraw** (hDc As OLE_HANDLE, nRow As CURRENCY, pColumn As ColumnHeader, xPos As Single, yPos As Single, xSize As Single, ySize As Single)

C++

void **OnDraw**(OLE_HANDLE hDc, CURRENCY nRow, ColumnHeader* pColumn, long xPos, long yPos, long xSize, long ySize)

The OnDraw event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
hDc	A handle to a Windows device context in which the drawing occurs.
nRow	The row that contains the cell that is being drawn.
pColumn	The column that contains the cell that is being drawn.
xPos	The x-origin of the cell that is being drawn.
yPos	The y-origin of the cell that is being drawn.
xSize	The width of the cell that is being drawn.
ySize	The height of the cell that is being drawn.

Remarks

This event allows a cell within a ColumnHeader with its OwnerDraw property set to TRUE to perform its custom drawing.

Example

C++

```
void CInaGridContainer::OnDraw(long hDc, CURRENCY nRow, LPDISPATCH pColumn,
long xPos, long yPos, long xSize, long ySize)
{
    CBitmap bmpDraw;
    bmpDraw.LoadBitmap(IDB_BITMAP);

    CDC* pDC = CDC::FromHandle((HDC)hDc);
    CDC dcBitmap;
    dcBitmap.CreateCompatibleDC(pDC);
    CBitmap* pBitmap = dcBitmap.SelectObject(&bmpDraw);
    pDC->BitBlt(xPos, yPos, xSize, ySize, &dcBitmap, 0, 0, SRCCOPY);
    dcBitmap.SelectObject(pBitmap);
    bmpDraw.DeleteObject();
}
```

OnEditCell Event

See Also FAQ Properties Methods Events Enums

Occurs when the user initiates a cell edit.

Syntax

VB

Private Sub **object_OnEditCell** (nRow As CURRENCY, pColumn As ColumnHeader, pClassID As String)

C++

void **OnEditCell**(CURRENCY nRow, LPDISPATCH pColumn, BSTR FAR* pClassId);

The OnEditCell event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
nRow	The row that contains the cell that is being edited.
pColumn	The column that contains the cell that is being edited.
pClassID	The CLSID of the editing control that performs the cell edit.

Remarks

The user initiates an edit action by double-clicking on the cell with the mouse or pressing the F2 key. If an InaGrid cell has the focus, pressing ENTER sets the first editable cell on the row into edit mode. Subsequent presses of the TAB key moves through all of the editable cells in the row in turn, placing each one into edit mode.

The pClassID parameter is used to select the editing control for the cell. By default, the *InaEdit* control is used. If an empty string is returned, no editing is performed in the cell.

It is not necessary to catch this event if the default edit box control is used for data editing.

An arbitrary valid Class ID can be specified for the *pClassID* parameter. The InaGrid control object will create this OCX edit control, and will invoke the *OnInitEditCell* event so that this edit control can be initialized. If the edit process finishes successfully the InaGrid control object will get the text of the edit control and will invoke the *OnSetData* event to notify the container so that the application can save the user's entry data. The InaGrid control object works with the standard OCX property "Caption" of this control both to initialize the text before editing and to extract the user's entry data after editing.

NOTE: See the InaGrid Editing Controls:

- InaEdit
- InaCombo
- InaCheck

Example

VB

```
Private Sub GridEvts_OnEditCell(ByVal nRow As INAGRIDLib.CURRENCY, ByVal
pColumn As INAGRIDLib.IColumnHeader, pClassId As String)

    If Not pColumn Is Nothing Then

        Select Case pColumn.Id

        Case 0
            ' Default InaEdit control

        Case 1
            'return InaCombo ClassID
            pClassId = "{0052B91E-9C90-11D1-B46C-0080ADC8C04D}"

        Case 2
```

```
            'return InaCheck ClassID
            pClassId = "{3844F267-9D42-11D1-B46C-0080ADC8C04D}"

        Case 3
            'return the ClassID of your OCX control
            pClassId = "{########-####-####-####-############}"

        Case Else
            'disable editing
            pClassId = ""

        End Select

    End If

End Sub
```

C++

```
void CInaGridContainer::OnEditCell(CURRENCY nRow, LPDISPATCH pColumn, BSTR
FAR* pClassId)
{
    if (NULL != pColumn)
    {
        CColumnHeader hdrColumn(pColumn);

        switch(hdrColumn.GetId())
        {
        case 0:
            // edit box--leave default
            break;

        case 1:
            // combo box
            SysFreeString(*pClassId);

            // return InaCombo ClassID
            *pClassId = SysAllocString(L"{0052B91E-9C90-11D1-B46C-
0080ADC8C04D}")
            break;

        case 2:
            // check box
            SysFreeString(*pClassId);

            // return InaCheck ClassID
            *pClassId = SysAllocString(L"{3844F267-9D42-11D1-B46C-
0080ADC8C04D}");
            break;

        case 3:
            // user OCX control
            SysFreeString(*pClassId);

            // Return the ClassID of your OCX control
           *pClassId = SysAllocString(L"{########-####-####-####-
############}")
            break;

        default: // don't allow editing
            SysFreeString(*pClassId);
```

```
            *pClassId = NULL;
            break;
        }
        hdrColumn.DetachDispatch();
    }
}
```

OnInitEditCell Event

Occurs when the user initiates a cell edit after OnEditCell event. Catch this event to initialize the OCX control specified in OnEditCell event.

Syntax

VB

Private Sub **object_OnInitEditCell** (nRow As CURRENCY, pColumn As ColumnHeader, pControl As Object, pExtraWidth As Long, pExtraHeight As Long)

C++

void **OnInitEditCell**(CURRENCY nRow, LPDISPATCH pColumn, LPDISPATCH pControl, long FAR* pExtraWidth, long FAR* pExtraHeight)

The OnInitEditCell event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
nRow	The row that contains the cell that is being edited.
pColumn	The column that contains the cell that is being edited.
pControl	A pointer to the IDispatch interface on the edit control.
pExtraWidth	The width in pixels to add to the right-hand side of the InaGrid cell, beyond the normal cell boundary.
pExtraHeight	The height in pixels to add to the bottom side of the InaGrid cell, beyond the normal cell boundary.

Remarks

It is not necessary to catch this event if the edit cell requires no special initialization, i.e., default edit box control.

Example

VB

```
Private Sub GridEvts_OnInitEditCell(ByVal nRow As INAGRIDLib.CURRENCY, ByVal
pColumn As INAGRIDLib.IColumnHeader, ByVal pControl As Object, pExtraWidth As
Long, pExtraHeight As Long)

    Dim editHwnd As OLE_HANDLE
    Dim lResult As Long

    If Not pColumn Is Nothing Then

        Select Case pColumn.Id
```

```
        Case 0
            'EditBox--leave default

        Case 1

            'Combobox
            editHwnd = pControl.hWnd
            lResult = SendMessage(editHwnd, CB_ADDSTRING, 0, "Choice 1")
            lResult = SendMessage(editHwnd, CB_ADDSTRING, 0, "Choice 2")
            lResult = SendMessage(editHwnd, CB_SETCURSEL, 1, 0)

            pExtraHeight = GridControl1.RowHeight * 2

        Case 2
            'CheckBox
            editHwnd = pControl.hWnd
            pControl.Init "Choose", "Yes", "No"
            lResult = SendMessage(editHwnd, BM_SETCHECK, 0, 0)

        Case 3
            'Do the necessary initialization of your OCX control

        End Select

    End If

End Sub
```

C++

```
void CInaGridContainer::OnInitEditCell(CURRENCY nRow, LPDISPATCH pColumn,
LPDISPATCH pControl, long FAR* pExtraWidth, long FAR* pExtraHeight)
{
    if (NULL != pColumn)
    {
        CColumnHeader hdrColumn(pColumn);

        switch(hdrColumn.GetId())
        {
        case 0:
            // edit box--leave default
            break;

        case 1:
            //combo box
            {
                _DInaCombo inaCombo(pControl);
                ::SendMessage((HWND)inaCombo.GetHWnd(), CB_ADDSTRING, NULL,
(LPARAM)"Choice 1");
                ::SendMessage((HWND)inaCombo.GetHWnd(), CB_ADDSTRING, NULL,
(LPARAM)"Choice 2");
                ::SendMessage((HWND)inaCombo.GetHWnd(), CB_SETCURSEL,
(WPARAM)1, NULL);
              // extra height for the drop-down list
                *pExtraHeight = abs(m_wndGrid.GetRowHeight()) * 2;
                inaCombo.DetachDispatch();
            }
            break;

        case 2:
            // check box
            {
                _DInaCheck inaCheck(pControl);
```

```
                inaCheck.Init("Choose", "Yes", "No");
                ::SendMessage((HWND)inaCheck.GetHWnd(), BM_SETCHECK,
(WPARAM)0, NULL);
                inaCheck.DetachDispatch();
            }
            break;

        case 3:
            // user OCX control
            {
                // Do the necessary initialization of your OCX control
            }
            break;
        }
        hdrColumn.DetachDispatch();
    }
}
```

OnMoveColumn Event

Occurs when a column header is moved (drag & drop).

Syntax

VB

Private Sub object_**OnMoveColumn**(pColumn As ColumnHeader)

C++

void **On****MoveColumn**(LPDISPATCH pColumn);

The OnMoveColumn event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
pColumn	The column that is being moved.

Example

C++

```
void CInaGridContainer::OnMoveColumn(LPDISPATCH pColumn)
{
      //Center InaGrid control
      CRect rWnd;
      GetClientRect(rWnd);
      CY cyGridHeight = m_wndGrid.GetHeight();
      CY cyGridWidth = m_wndGrid.GetWidth();
      CRect rGrid(rWnd);
      if (cyGridHeight.int64 < rWnd.Height())
        rGrid.InflateRect(0, (int(cyGridHeight.Lo) - rWnd.Height()) / 2);
      if (cyGridWidth.int64 < rWnd.Width())
      rGrid.InflateRect((int(cyGridWidth.Lo) - rWnd.Width()) / 2, 0);
      rGrid.right = min(rGrid.left + cyGridWidth.Lo, rWnd.right);
```

```
      rGrid.bottom = min(rGrid.top + cyGridHeight.Lo, rWnd.bottom);
      m_wndGrid.MoveWindow(rGrid);
    }
```

OnResizeColumn Event

Occurs when a column header is resized.

Syntax

VB

Private Sub object_**OnResizeColumn**(pColumn As ColumnHeader)

C++

void **OnResizeColumn**(LPDISPATCH pColumn);

The OnResizeColumn event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
pColumn	The column that is being resized.

Example

```
void CInaGridContainer::OnResizeColumn(LPDISPATCH pColumn)
{
       //Center InaGrid control
       CRect rWnd;
       GetClientRect(rWnd);
       CY cyGridHeight = m_wndGrid.GetHeight();
       CY cyGridWidth = m_wndGrid.GetWidth();
       CRect rGrid(rWnd);
       if (cyGridHeight.int64 < rWnd.Height())
          rGrid.InflateRect(0, (int(cyGridHeight.Lo) - rWnd.Height()) / 2);
       if (cyGridWidth.int64 < rWnd.Width())
          rGrid.InflateRect((int(cyGridWidth.Lo) - rWnd.Width()) / 2, 0);
       rGrid.right = min(rGrid.left + cyGridWidth.Lo, rWnd.right);
       rGrid.bottom = min(rGrid.top + cyGridHeight.Lo, rWnd.bottom);
       m_wndGrid.MoveWindow(rGrid);
}
```

OnSelect Event

Occurs when a cell or row is selected.

Syntax

VB

Private Sub **object_OnSelect** (nRow As CURRENCY, pColumn As ColumnHeader, eType As GridSelectConstants)

C++

void **OnSelect**(CURRENCY nRow, LPDISPATCH pColumn, long eType);

The OnSelect event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
nRow	The row that is being selected.
pColumn	The column that is being selected.
eType	The type of selection that is being made.

Settings

The settings for *eType* are:

Setting	Description
gSingleSelect	A single selection is being made.
gSelectRange	A range of selections is being made using the Shift key.
gSelectToggle	Multiple selections are being made using the Control key.

Remarks

This event is called first when a selection is made by the user. Typically the application will invalidate the affected rows/cells and move the focus rectangle to the selected cell using *FocusRow* or *Focus/ColumnHeader*.

The values of GridSelectConstants are 0, 1 and 2 respectively for gSingleSelect, gSelectRange and gSelectToggle.

Example

VB

```
Private Sub GridEvts_OnSelect(ByVal nRow As INAGRIDLib.CURRENCY, ByVal pColumn
As INAGRIDLib.IColumnHeader, ByVal eType As INAGRIDLib.GridSelectConstants)
    Dim oldSelectedRow As CURRENCY
    oldSelectedRow = gSelectedRow
    gSelectedRow = nRow
    GridControl1.UpdateRow nRow
    GridControl1.FocusRow = nRow
    nRow = oldSelectedRow
    GridControl1.UpdateRow nRow

    gSelectedColumn = pColumn.Id
    GridControl1.FocusColumnHeader = pColumn

End Sub
```

C++

```
void CInaGridContainer::OnSelect(CURRENCY nRow, LPDISPATCH pColumn, long
eType)
{
    if(NULL != pColumn)
    {
        CURRENCY OldSelectedRow = m_n64SelectedRow;
        m_n64SelectedRow = nRow.int64;
        m_wndGrid.UpdateRow(nRow);
        m_wndGrid.SetFocusRow(nRow);
```

```
        nRow.int64 = n64OldSelectedRow;
        m_wndGrid.UpdateRow(nRow);

        CColumnHeader hdrColumn(pColumn);
        m_nSelectedColumn = hdrColumn.GetId();
        m_wndGrid.SetFocusColumnHeader(pColumn);
        hdrColumn.DetachDispatch();
    }
}
```

SetData Event

Occurs when a data in a cell is changed in the InaGrid control.

Syntax

VB

Private Sub **object_SetData**(nRow As CURRENCY, pColumn As ColumnHeader, pValue As String)

C++

void **OnSetData**(CURRENCY nRow, LPDISPATCH pColumn, LPCTSTR pValue);

The GetData event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an InaGrid control.
nRow	The row that contains the changed data.
pColumn	The column that contains the changed data.
pValue	The changed data at the row and column.

Remarks

This event occurs when the data in a cell in the InaGrid control has been changed by an editing action. This allows the application to update its data structures to reflect the changed data.

ColumnHeaders Object

ColumnHeaders Object

The Inagrid control is a system of objects:

- GridControl (contains the ColumnHeaders object)
- ColumnHeaders (this object, which contains the ColumnHeader object)
- ColumnHeader
- Format Data (used for cell formatting)

- DataObjectFiles (used for drag and drop)
- DataObject (used for drag and drop)

The ColumnHeaders collection object provides properties and methods on the set of ColumnHeader objects that are part of an InaGrid control.

Properties

Count Property (ColumnHeaders Object)

Returns the number of ColumnHeader objects contained within the ColumnHeaders collection.

Syntax

VB

Object.**Count**

C++

long *Object*.**GetCount**()

Part	Description
Object	An object expression that evaluates to a ColumnHeaders collection object.

Remarks

This is a read-only property.

Example

VB

```
For i = 0 To GridControl1.ColumnHeaders.Count –1
    If GridControl1.ColumnHeaders(i).Id = SelectedColumn Then
        Set GridControl1.FocusColumnHeader = GridControl1.ColumnHeaders(i)
    End If
Next I
```

C++

```
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
int nHeaderCount = hdrCol.GetCount();
for(int nHeader = 0; nHeader < nHeaderCount; nHeader++)
{
    CColumnHeader header = hdrCol.GetItem(nHeader);
    if(header.GetId() == nId)
        return nHeader;
}
```

Item Property (ColumnHeaders Object)

Returns a ColumnHeader object contained within the ColumnHeaders collection.

Syntax

VB

Object.Item(nIndex As Long)

C++

CColumnHeader *Object.***GetItem**(long nIndex)

The Item property syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeaders collection object.
nIndex	The 0-based index position of the ColumnHeader object.

Remarks

The ColumnHeaders collection is a 0-based collection.

Example

VB

```
Dim aColHdrs As ColumnHeaders
Set aColHdrs = GridControl1.ColumnHeaders
aColHdrs.Item(1).Width = 125
```

C++

```
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
int nHeaderCount = hdrCol.GetCount();
for(int nHeader = 0; nHeader < nHeaderCount; nHeader++)
{
    CColumnHeader header = hdrCol.GetItem(nHeader);
    if(header.GetId() == nId)
        return nHeader;
}
```

Parent Property (ColumnHeaders Object)

Returns or sets the InaGrid control object that the ColumnHeaders collection belongs to.

Syntax

VB

object.**Parent**

C++

LPDISPATCH *Object.***GetParent**();

The Parent property syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeaders collection object.
parentObject	An object expression that evaluates to an InaGrid control object.

Example

VB

```
Dim aGridControl As Object
Set aGridControl = GridControl1.ColumnHeaders.Parent
```

C++

```
LPDISPATCH pGridDisp = hdrCol.GetParent();
CGridControl* pGridControl =
(CGridControl*)CCmdTarget::FromIDispatch(pGridDisp);
```

Methods

Add Method (ColumnHeaders Object)

Adds and returns a new ColumnHeader object to the ColumnHeaders collection.

Syntax

VB

*object.***Add(**nIndex As Long, pText As String, nWidth As OLE_XSIZE_PIXELS, nId As Long, eHAlignment As GridColumnHAlignmentConstants, eVAlignment As GridColumnVAlignmentConstants) As ColumnHeader

C++

CColumnHeader *Object.***Add**(long nIndex, LPCTSTR pText, long nWidth, long nId, long eHAlignment, long eVAlignment)

The Add method syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeaders collection object.
nIndex	0-based position within the ColumnHeaders collection in which to insert the new ColumnHeader.
pText	The column header text.
nWidth	The width of the column in pixels.
nId	The unique Id to assign to the column.
eHAlignment	The horizontal alignment of the text that is displayed in

	the column.
eVAlignment	The vertical alignment of the text that is displayed in the column.

Settings

The settings for eHAlignment are:

Setting	Description
gColumnLeft	The sub-header text is displayed left-aligned.
gColumnCenter	The sub-header text is displayed center-aligned.
gColumnRight	The sub-header text is displayed right-aligned.

The settings for eVAlignment are:

Setting	Description
gColumnTop	The column text is displayed top-aligned.
gColumnVCenter	The column text is displayed vertically center-aligned.
gColumnBottom	The column text is displayed bottom-aligned.

Remarks

This method adds new ColumnHeader objects at the position specified in *nIndex*. If *nIndex* is specified with the value of –1, a new ColumnHeader object is added at the end of the list.

Adding a new column header object resizes all column headers in sub-header rows which span the position where the header is added.

See also : Alignment Properties (ColumnHeader Object)

Example

VB

```
GridControl1.ColumnHeaders.Add -1, "First Name", 100, 0, gColumnLeft,
gColumnBottom;
GridControl1.ColumnHeaders.Add -1, "Last Name", 150, 1, gColumnLeft,
gColumnBottom;
```

C++

```
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
hdrCol.Add(-1, "First Name", 100, 0, gColumnLeft), gColumnBottom;
hdrCol.Add(-1, "Last Name", 150, 1, gColumnLeft), gColumnBottom;
```

AddSubHeader Method (ColumnHeaders Object)

Adds and returns a new ColumnHeader object that spans one or more columns.

Syntax

VB

object.**AddSubHeader(**nSubHeader As Long, nFirstCol As Long, nLastCol As Long, pText As String, nId As Long, eHAlignment As GridColumnHAlignmentConstants, eVAlignment As GridColumnVAlignmentConstants) As ColumnHeader

C++

CColumnHeader *Object.***AddSubHeader**(long nSubHeader, long nFirstCol, long nLastCol, LPCTSTR pText, long nId, long eHAlignment, long eVAlignment)

The **AddSubHeader** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeaders collection object.
nSubHeader	The 0-based vertical position of the sub-header. See remarks.
nFirstCol	The first 0-based column position that the sub-header is positioned above.
nLastCol	The last 0-based column position that the sub-header is positioned above.
pText	The sub-header text.
nId	The unique Id to assign to the sub-header.
eHAlignment	The horizontal alignment of the text that is displayed in the sub-header.
eVAlignment	The vertical alignment of the text that is displayed in the sub-header.

Settings

The settings for eHAlignment are:

Setting	Description
gColumnLeft	The sub-header text is displayed left-aligned.
gColumnCenter	The sub-header text is displayed center-aligned.
gColumnRight	The sub-header text is displayed right-aligned.

The settings for eVAlignment are:

Setting	Description
gColumnTop	The sub-header text is displayed top-aligned.
gColumnVCenter	The sub-header text is displayed vertically center-aligned.
gColumnBottom	The sub-header text is displayed bottom-aligned.

Remarks

The nSubHeader index specifies how the subheaders are placed in relation to each other vertically. The first sub-header position, nSubHeader = 0, is placed at the top most position of the headers. Since sub headers can span multiple columns, those that are intended to be placed side by side should have the same nSubHeader value.

Do not add a sub-header to a row that spans a column in another sub-header in the same row. For example if there is a column header in a sub-header row with first and last columns set to 2 and 5, you cannot add a new column header in the same sub-header with first and last columns set to 4 and 7 because columns 4 and 5 would belong to both column headers.

See also : Alignment Properties (ColumnHeader Object)

Example

VB

```
GridControl1.ColumnHeaders.AddSubHeader 0, 0, 5, "Personal Data", 6,
gColumnCenter, gColumnVCenter
GridControl1.ColumnHeaders.AddSubHeader 1, 0, 1, "Name", 7, gColumnCenter,
gColumnVCenter
GridControl1.ColumnHeaders.AddSubHeader 1, 2, 5, "Attributes", 8,
gColumnCenter, gColumnVCenter
```

C++

```
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
hdrCol.AddSubHeader(0, 0, 5, "Personal Data", 6, gColumnCenter),
gColumnVCenter;
hdrCol.AddSubHeader(1, 0, 1, "Name", 7, gColumnCenter), gColumnVCenter;
hdrCol.AddSubHeader(1, 2, 5, "Attributes", 8, gColumnCenter), gColumnVCenter;
```

Clear Method (ColumnHeaders Object)

Removes all ColumnHeader objects and sub-headers from the ColumnHeaders collection.

Syntax

VB

object.**Clear**

C++

void *Object*.**Clear**()

The **Clear** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeaders collection object.

Example

VB

```
Dim aColHdrs As ColumnHeaders
Set aColHdrs = GridControl1.ColumnHeaders
aColHdrs.Clear
```

C++

```
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
hdrCol.Clear();
```

ClearSubHeaders Method (ColumnHeaders Object)

Removes all sub-headers from the ColumnHeaders collection.

Syntax

VB

object.**ClearSubHeaders**()

C++

void *Object.* **ClearSubHeaders**()

The **ClearSubHeaders** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeaders collection object.

Example

VB

```
GridControl1.ColumnHeaders.ClearSubHeaders
```

C++

```
CColumnHeaders hdrCols = m_wndGrid.GetColumnHeaders();
hdrCols.ClearSubHeaders();
```

GetSubHeader Method (ColumnHeaders Object)

Returns a ColumnHeader object that represents a sub-header.

Syntax

VB

object.**GetSubHeader(**nSubHeader As Long, nColumn As Long) As ColumnHeader

C++

CColumnHeader *Object.* **GetSubHeader**(long nSubHeader, long nColumn)

The **GetSubHeader** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeaders collection object.
nSubHeader	The 0-based vertical position of the sub-header. See remarks.
nColumn	The 0-based index of the sub-header.

Remarks

The nSubHeader index specifies how the subheaders are placed in relation to each other vertically. The first sub-header position, nSubHeader = 0, is placed at the top most position of the headers. Since sub headers can span multiple columns, those that are intended to be placed side by side should have the same nSubHeader value.

Example

VB

```
'Get "Attributes" sub-header
Dim aColHdr As ColumnHeader
Set aColumnHeader = GridControl1.ColumnHeaders.GetSubHeader (1,1)
```

C++

```
// Get "Attributes" sub-header
CColumnHeaders hdrCols = m_wndGrid.GetColumnHeaders();
CColumnHeader aColHdr = hdrCols.GetSubHeader(1,1);
```

GetSubHeaderColumnCount Method (ColumnHeaders Object)

Returns the number of sub-headers on a specified sub-header row.

Syntax

VB

object.**GetSubHeaderColumnCount**(nSubHeader As Long) As Long

C++

long *Object.* **GetSubHeaderColumnCount**(long nSubHeader)

The **GetSubHeaderColumnCount** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeaders collection object.
nSubHeader	The 0-based vertical position of the sub-header. See remarks.

Remarks

The nSubHeader index specifies how the subheaders are placed in relation to each other vertically. The first sub header position, nSubHeader = 0, is placed at the top most position of the headers. Since sub headers can span multiple columns, those that are intended to be placed side by side should have the same nSubHeader value.

Example

VB

```
'Get number of columns that the first sub-header position spans
Dim lSubColCount As Long
lSubColCount = GridControl1.ColumnHeaders.GetSubHeaderColumnCount(0)
```

C++

```
// Get number of columns that the first sub-header position spans
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
long lSubColCount = hdrCol.GetSubHeaderColumnCount(0);
```

GetSubHeaderLineCount Method (ColumnHeaders Object)

Returns the number of rows that contain sub-headers.

Syntax

VB

object.GetSubHeaderLineCount() As Long

C++

long *Object.* GetSubHeaderLineCount()

The **GetSubHeaderLineCount** method syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeaders collection object.

Example

VB

```
Dim lSubCount As Long
lSubCount = GridControl1.ColumnHeaders.GetSubHeaderLineCount()
```

C++

```
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
long lSubCount = hdrCol.GetSubHeaderLineCount();
```

Remove Method (ColumnHeaders Object)

Removes the specified ColumnHeader object from the ColumnHeaders collection.

Syntax

VB

object.**Remove** nIndex As Long

C++

void *Object.***Remove**(long nIndex)

The **Remove** method syntax has these parts:

Part	Description

Object	An object expression that evaluates to a ColumnHeaders collection object.
nIndex	The 0-based index of the ColumnHeader object to remove from the collection.

Remarks

If a column header is removed that is spanned by one or more sub-headers, all column headers from sub-header rows, which contain this column, will be resized. If the removed column is the only column they span, then the sub-header column is removed as well.

Example

VB

```
'Remove the fourth ColumnHeader object
Dim aColHdrs As ColumnHeaders
Set aColHdrs = GridControl1.ColumnHeaders
aColHdrs.Remove 3
```

C++

```
// Remove the fourth ColumnHeader object
CColumnHeaders hdrCol = m_wndGrid.GetColumnHeaders();
hdrCol.Remove(3);
```

ColumnHeader Object

ColumnHeader Object

See Also FAQ Properties Methods Events Enums

The InaGrid control is a system of objects:

- GridControl (contains the ColumnHeaders object)
- ColumnHeaders (contains the ColumnHeader object)
- ColumnHeader (this object)
- Format Data (used for cell formatting)
- DataObjectFiles (used for drag and drop)
- DataObject (used for drag and drop)

The ColumnHeader object describes a column in the InaGrid control. There is one ColumnHeader object for each column in the control. The set of ColumnHeader objects for an InaGrid control is contained within a ColumnHeaders collection object, which provides properties and methods on the set of ColumnHeader objects.

Properties

Alignment Properties (ColumnHeader Object)

See Also FAQ Properties Methods Events Enums

These are the ColumnHeader alignment properties:

- HAlignment--Returns or sets the horizontal alignment for data that is displayed in the column.
- VAlignment--Returns or sets the vertical alignment for data that is displayed in the column.

Syntax

VB

Object.**HAlignment** [= h*Align*]
Object.**VAlignment** [= v*Align*]

C++

long *Object.***GetHAlignment**()
void *Object.***SetHAlignment**(long hAlign)

*long Object.**GetVAlignment**()*
*void Object.**SetVAlignment**(long vAlign)*

Part	Description
Object	An object expression that evaluates to a ColumnHeader object.
hAlign	A GridColumnHAlignmentConstants enumerator specifying the type of horizontal alignment to use when displaying text data.
vAlign	A GridColumnVAlignmentConstants enumerator specifying the type of vertical alignment to use when displaying text data.

Settings

The settings for *hAlign* are:

Setting	Description
gColumnLeft	The column text is displayed left-aligned.
gColumnCenter	The column text is displayed center-aligned.
gColumnRight	The column text is displayed right-aligned.

The settings for *vAlign* are:

Setting	Description
gColumnTop	The column text is displayed at the top of the cell.
gColumnVCenter	The column text is displayed vertically centered.
gColumnBottom	The column text is displayed along the bottom of the cell.

Remarks

The vertical alignment settings have no effect if the InaGrid WrapColumnHeaders property is set to TRUE.

Example

VB

```
Dim aColHdrs As ColumnHeaders
```

```
Set aColHdrs = GridControl1.ColumnHeaders
aColHdrs.Item(1).HAlignment = gColumnCenter
aColHdrs.Item(1).VAlignment = gColumnTop
```

C++

```
CColumnHeader header = hdrCol.GetItem(1);
header.SetHAlignment(gColumnCenter);
header.SetVAlignment(gColumnTop);
```

FirstColumn, LastColumn Properties (ColumnHeader Object)

FirstColumn and LastColumn properties operate as follows:

- FirstColumn--Returns the id of the first InaGrid control column that a sub-header is positioned over.
- LastColumn--Returns the id of the last InaGrid control column that a sub-header is positioned over.

Syntax

VB

Object.**FirstColumn**

Object.**LastColumn**

C++

long *Object.***GetFirstColumn**()

*long Object.**GetLastColumn**()*

Part	**Description**
Object	An object expression that evaluates to a ColumnHeader object.

Remarks

These properties are used with sub-header column headers. When a column header represents a sub-header (spans over several columns) these values represent the starting and ending columns of the grid, over which this header spans. For example if a column header is an item of one of the sub-headers and it has FirstColumn and LastColumn properties set respectively to 5 and 7, this header will span over the 5-th, 6-th and the 7-th columns of the grid. These properties are not useful if the column header does not represent a sub-header.

These properties are read-only.

If no column exists in the first or last position, or if the column header object does not represent a sub-header, -1 is returned.

Example

VB

```
'Get "Attributes" sub-header
Dim aColHdr As ColumnHeader
Set aColumnHeader = GridControl1.ColumnHeaders.GetSubHeader (1,1)
```

```
lFirstCol = aColHdr.FirstColumn
lLastCol = aColHdr.LastColumn
```

C++

```
// Get "Attributes" sub-header
CColumnHeaders hdrCols = m_wndGrid.GetColumnHeaders();
CColumnHeader aColHdr = hdrCols.GetSubHeader(1,1);
long lFirstCol = aColHdr.GetFirstColumn();
long lLastCol = aColHdr.GetLastColumn();
```

FixedWidth Property (ColumnHeader Object)

Returns or sets whether the column width can be changed by the user.

Syntax

VB

Object.**FixedWidth** [= *bValue*]

C++

BOOL *Object.***GetFixedWidth**()

*void Object.****SetFixedWidth****(BOOL bValue)*

Part	Description
Object	An object expression that evaluates to a ColumnHeader object.
bValue	A Boolean expression specifying whether the user can resize the column.

Settings

The settings for bValue are:

Setting	Description
True	The user can resize the column by dragging the sizing icon at the edge of the column heading.
False	The column width remains fixed and cannot be changed.

Remarks

The FixedWidth property does not change the ability to modify the column width using the Width property.

Example

VB

```
Dim aCol As ColumnHeader
Set aCol = GridControl1.ColumnHeaders.Add(-1, "Second Column", 100, 2,
gColumnLeft), gColumnVCenter)
aCol.FixedWidth = True
```

C++

```
CColumnHeader header = hdrCol.Add(-1, "Second Column", 100, 2, gColumnLeft) ,
gColumnVCenter);
header.SetFixedWidth(TRUE);
```

Id Property (ColumnHeader Object)

Returns or sets the identification number of the column.

Syntax

VB

Object.**Id** [= number]

C++

long *Object.***GetId**()

*void Object.**SetId**(long number)*

Part	Description
Object	An object expression that evaluates to a ColumnHeader object.
Number	A value or constant that specifies the id of the column.

Remarks

This property is intended to represent an application-defined value that identifies the column. This is useful when responding to the *GetData* event which requests data for a particular row and column.

Example

VB

```
pValue = "Row = " + Str(nRow * 10000) + ", Col = " + Str(pColumn.Id)
```

C++

```
CColumnHeader hdrColumn(pColumn);
long nCol = hdrColumn.GetId();
```

OwnerDraw Property (ColumnHeader Object)

Returns or sets whether the data displayed in the column is drawn by the programmer.

Syntax

VB

Object.**OwnerDraw** [= *bValue*]

C++

BOOL *Object.***GetOwnerDraw**()

*void Object.**SetOwnerDraw**(BOOL bValue)*

Part	Description
Object	An object expression that evaluates to a ColumnHeader object.
bValue	A Boolean expression specifying whether data is drawn by the programmer.

Settings

The settings for bValue are:

Setting	Description
True	The programmer is responsible for drawing the data presentation. The programmer must respond to the OnDraw event and use Windows functions to display the data.
False	The InaGrid control performs all data display functionality.

Remarks

Setting the OwnerDraw property to TRUE allows the programmer to present text or graphic data for the column using Windows text and GDI routines.

Example

VB

```
Dim aCol As ColumnHeader
Set aCol = GridControl1.ColumnHeaders.Add(-1, "Second Column", 100, 2,
gColumnLeft), gColumnVCenter)
aCol.OwnerDraw = True
```

C++

```
CColumnHeader header = hdrCol.Add(-1, "Second Column", 100, 2, ColumnLeft),
gColumnVCenter);
header.SetOwnerDraw(TRUE);
```

Parent Property (ColumnHeader Object)

Returns or sets the ColumnHeaders container for the ColumnHeader object.

Syntax

VB

Object**.Parent** [= columnHeaders]

C++

CColumnHeaders *Object.***GetParent**()

*void Object.**SetParent**(LPDISPATCH columnHeaders)*

Part	Description
Object	An object expression that evaluates to a ColumnHeader object.
columnHeaders	Specifies the ColumnHeaders object that serves as the container for the column header.

Selected Property (ColumnHeader Object)

Returns or sets the selection state of the column.

Syntax

VB

Object.**Selected** [= *bValue*]

C++

BOOL *Object.***GetSelected**()

*void Object.**SetSelected**(BOOL bValue)*

Part	Description
Object	An object expression that evaluates to a ColumnHeader object.
bValue	Specifies whether the column is selected.

Settings

The settings for bValue are:

Setting	Description
True	The column is selected. The cell containing the selected row is painted in the highlighted colors.
False	The column is not selected.

Example

VB

```
Dim aColHdrs As ColumnHeaders, bSelected As Boolean
Set aColHdrs = GridControl1.ColumnHeaders
bSelected = aColHdrs.Item(1).Selected
```

C++

```
CColumnHeader header = hdrCol.GetItem(1);
BOOL bSelected = header.GetSelected();
```

Text Property (ColumnHeader Object)

Returns or sets the text displayed in the column header.

Syntax

VB

object.**Text** [= string]

C++

CString *Object.***GetText**()

void *Object.***SetText**(LPCTSTR string)

The Text property syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeader object.
string	A string expression specifying the text appearing in column header.

Example

VB

```
Dim aColHdrs As ColumnHeaders
Set aColHdrs = GridControl1.ColumnHeaders
aColHdrs.Item(1).Text = "New column text"
```

C++

```
CColumnHeader header = hdrCol.GetItem(1);
header.SetText("New column text");
```

Width Property (Column Header Object)

Returns or sets the width of the column.

Syntax

VB

Object.**Width** [= lVal]

C++

long *Object.***GetWidth**()

void *Object.***SetWidth**(long lVal)

The Width property syntax has these parts:

Part	Description
Object	An object expression that evaluates to a ColumnHeader object.
lVal	A value or constant that determines the width of the column.

Example

VB

```
Dim aColHdrs As ColumnHeaders
Set aColHdrs = GridControl1.ColumnHeaders
aColHdrs.Item(1).Width = 125
```

C++

```
CColumnHeader header = hdrCol.GetItem(1);
header.SetWidth(125);
```

Format Data Object (for Cell Formatting)

FormatData Object

The InaGrid control is a system of objects:

- GridControl (contains the ColumnHeaders object)
- ColumnHeaders (contains the ColumnHeader object)
- ColumnHeader
- Format Data (this object)
- DataObjectFiles (Used for drag and drop)
- DataObject (Used for drag and drop)

The FormatData object describes how a cell is formatted in the InaGrid control.

Properties

Alignment Properties (FormatData Object)

FormatData alignment properties :

- HorizontalAlignment--Returns or sets the horizontal alignment for data that is displayed in the cell.
- VerticalAlignment--Returns or sets the vertical alignment for data that is displayed in the cell.

Syntax

VB

Object.**HorizontalAlignment** [= h*Align*]
Object.**VerticalAlignment** [= v*Align*]

C++

long *Object.***GetHorizontalAlignment**()
void *Object.***SetHorizontalAlignment**(long hAlign)

long *Object.***GetVerticalAlignment**()
void *Object.***SetVerticalAlignment**(long vAlign)

Part	Description
Object	An object expression that evaluates to a FormatData object.
hAlign	A GridColumnHAlignmentConstants enumerator specifying the type of horizontal alignment to use when displaying text data.
vAlign	A GridColumnVAlignmentConstants enumerator specifying the type of vertical alignment to use when displaying text data.

Settings

The settings for *hAlign* are:

Setting	Description
gColumnLeft	The cell text is displayed left-aligned.
gColumnCenter	The cell text is displayed center-aligned.
gColumnRight	The cell text is displayed right-aligned.

The settings for *vAlign* are:

Setting	Description
GColumnTop	The cell text is displayed at the top of the cell.
GcolumnVCenter	The cell text is displayed vertically centered.
GcolumnBottom	The cell text is displayed along the bottom of the cell.

Remarks

The vertical alignment settings have no effect if the InaGrid WrapColumnHeaders property is set to TRUE.

Example

VB

```
aFormatData.HorizontalAlignment = gColumnCenter
aFormatData.VerticalAlignment = gColumnTop
```

C++

```
pFormatData->SetHorizontalAlignment(gColumnCenter);
pFormatData->SetVerticalAlignment(gColumnTop);
```

BackColor Property (FormatData Object)

Returns or sets the background color of rows in an InaGrid control.

Syntax

VB

Object.***BackColor*** [= color]

C++

OLE_COLOR *Object*.**GetBackColor** ()

void Object.***SetBackColor*** *(OLE_COLOR color)*

Part	Description
Object	An object expression that evaluates to a FormatData object.
Color	A value or constant that determines the background and foreground colors of an object, as described in Settings.

Settings

The InaGrid control uses the Microsoft Windows operating environment red-green-blue (RGB) color scheme. The settings for color are:

Setting	Description
Normal RGB colors	Colors specified by using the Color palette or by using the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VB) object library in the Object Browser . The Windows operating environment substitutes user choices as specified in the Control Panel settings.

Remarks

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the Visual Basic (VB) object library in the Object Browser.

To display text in the Windows operating environment, both the text and background colors must be solid. If the text or background colors you've selected aren't displayed, one of the selected colors may be dithered--that is, comprised of up to three different-colored pixels. If you choose a dithered color for either the text or background, the nearest solid color will be

substituted.

See Visual C++ Help. If MSB of color set, then color contains index used in GetSysColor: 0x800000xx, xx is a valid GetSysColor index.

Example

VB

```
GridControl1.BackColor = "&H00FF00"
GridControl1.TextColor = "&H0000FF"
```

C++

```
m_wndGrid.SetTextColor(RGB(0x00, 0xff, 0x00));
m_wndGrid.SetBackColor(RGB(0x00, 0x00, 0xff));
```

Font Properties (FormatData Object)

Returns or sets the font displayed in the cell.

- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeThrough
- FontCharSet

Syntax

VB

Object.**FontName** [= c*Value*]
Object.**FontSize** [= l*Value*]
Object.**FontBold** [= b*Value*]
Object.**FontItalic** [= b*Value*]
Object.**FontUnderline** [= b*Value*]
Object.**FontStrikeThrough** [= b*Value*]
Object.**FontCharSet** [= l*Value*]

C++

BSTR *Object.***GetFontName**()
void *Object.***SetFontName**(LPCTSTR cValue)

long *Object.***GetFontSize**()
void *Object.***SetFontSize**(LPCTSTR lValue)

BOOL *Object.***GetFontBold**()
void *Object.***SetFontBold**(BOOL bValue)

BOOL *Object.***GetFontItalic**()
void *Object.***SetFontItalic**(BOOL bValue)

BOOL *Object.***GetFontUnderline**()
void *Object.***SetFontUnderline**(BOOL bValue)

BOOL *Object.***GetFontStrikeThrough**()
void *Object.***SetFontStrikeThrough** (BOOL bValue)

*long Object.**GetFontCharSet**()*
*void Object.**SetFontCharSet**(long lValue)*

Part	Description
Object	An object expression that evaluates to an FormatData control.
cValue	A string defined in the Visual Basic StdType library.
bValue	A boolean value
lValue	A number

Remarks

Use the Font properties to identify a specific Font whose properties you want to use. For example, the following code changes the Bold property setting of a FormatData object identified by the GetFormat event of an InaGrid control:

Example

VB

```
FormatData.FontBold = False
```

C++

```
pFormatData->SetFontBold(FALSE);
```

TextColor Property (FormatData Object)

Returns or sets the background color of rows in an InaGrid control.

Syntax

VB

*Object.**TextColor*** [= color]

C++

OLE_COLOR *Object.***GetTextColor** ()

*void Object.**SetTextColor** (OLE_COLOR color)*

Part	Description
Object	An object expression that evaluates to a FormatData

	object.
Color	A value or constant that determines the background and foreground colors of an object, as described in Settings.

Settings

The InaGrid control uses the Microsoft Windows operating environment red-green-blue (RGB) color scheme. The settings for color are:

Setting	Description
Normal RGB colors	Colors specified by using the Color palette or by using the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VB) object library in the Object Browser . The Windows operating environment substitutes user choices as specified in the Control Panel settings.

Remarks

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the Visual Basic (VB) object library in the Object Browser.

To display text in the Windows operating environment, both the text and background colors must be solid. If the text or background colors you've selected aren't displayed, one of the selected colors may be dithered--that is, comprised of up to three different-colored pixels. If you choose a dithered color for either the text or background, the nearest solid color will be substituted.

See Visual C++ Help. If MSB of color set, then color contains index used in GetSysColor: 0x800000xx, xx is a valid GetSysColor index.

Example

VB

```
GridControl1.BackColor = "&H00FF00"
GridControl1.TextColor = "&H0000FF"
```

C++

```
m_wndGrid.SetTextColor(RGB(0x00, 0xff, 0x00));
m_wndGrid.SetBackColor(RGB(0x00, 0x00, 0xff));
```

WrapText Property (FormatData Object)

See Also FAQ Properties Methods Events Enums

Returns or sets the text wrapping for a cell.

Syntax

VB

Object.**WrapText** [= bValue]

C++

BOOL *Object.***GetWrapText**()
void *Object.***SetWrapText**(BOOL bValue)

Part	Description
Object	An object expression that evaluates to a FormatData object.
bValue	A boolean value

Example

VB

```
aFormatData.WrapText = False
```

C++

```
pFormatData->SetWrapText(FALSE);
```

VBData ObjectFiles (for Drag and Drop)

DataObjectFiles Object

See Also FAQ Properties Methods Events Enums

The InaGrid control is a system of objects:

- GridControl (contains the ColumnHeaders object)
- ColumnHeaders (contains the ColumnHeader object)
- ColumnHeader
- Format Data (Used for cell formatting)
- DataObjectFiles (this object)
- DataObject (Used for drag and drop)

The DataObjectFiles object describes the File list for Drag and Drop. A collection whose elements represent a list of all filenames used by a DataObject object.

Syntax

object.**DataObjectFiles**(index)

The DataObjectFiles collection syntax has these parts:

Part	Description
Object	An object expression that evaluates to a DataObjectFiles object.
Index	An integer with a range from 0 to DataObjectFiles.Count - 1.

Note: This collection is used by the Files property only when the data in the DataObject object is in the vbCFFiles format.

The DataObjectFiles collection is used by the Files property to store filenames in a DataObject object. It includes the Remove, Add, and Clear methods which allow you to manipulate its contents.

Methods

Add Method (DataObjectFiles Object)

Adds a file to a DataObjectFiles object.

Syntax

object.**Add file**, index

The Add Method syntax has these parts:

Part	Description
Object	An object expression that evaluates to a DataObjectFiles object.
File	A string that represents the file.
Index	An integer that identifies a member of the object collection. If supplied, the new member will be inserted after the member specified by the index.

Clear Method (DataObjectFiles Object)

Removes all objects in DataObjectFiles object.

Syntax

object.**Clear**

The object placeholder represents an object expression that evaluates to a DataObjectFiles object.

Remarks

To remove only one object from a collection, use the Remove method.

GetCount Method (DataObjectFiles Object)

Returns the number of objects in the DataObjectFiles object.

Syntax

object.**GetCount**

The Count property syntax has these parts:

Part	Description
Object	Required. A valid DataObjectFiles object.

GetItem Method (DataObjectFiles Object)

Returns a specific member of the DataObjectFiles object either by position or by key.

Syntax

object.**GetItem**(index)

The Item property syntax has the following object qualifier and part:

Part	Description
Object	Required. An object expression that evaluates to a DataObjectFiles object.
Index	Required. An expression that specifies the position of a member of the collection. If a numeric expression, index must be a number from 1 to the value of the collection's Count property. If a string expression, index must correspond to the key argument specified when the member referred to was added to the collection.

Remarks

If the value provided as index doesn't match any existing member of the collection, an error occurs.

NewEnum Method (DataObjectFiles Object)

References objects in DataObjectFiles.

Syntax

object._NewEnum

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.

Remarks

With Visual C++, you can browse a DataObjectFiles object to find a particular item by using the _NewEnum property. In Visual Basic, you do not need to use the _NewEnum property, because it is automatically used in the implementation of For Each ... Next.

Remove Method (DataObjectFiles Object)

Removes a specific member from a DataObjectFiles collection.

Syntax

object.**Remove** index

The Remove method syntax has these parts:

Part	Description
Object	An object expression that evaluates to a DataObjectFiles object.
Index	An integer or string that uniquely identifies the object within the collection. Use an integer to specify the value of the Index property; use a string to specify the value of the Key property.

Remarks

To remove all the members of a collection, use the Clear method.

VBDataObject (for Drag and Drop)

DataObject Object

The InaGrid control is a system of objects:

- GridControl (contains the ColumnHeaders object)
- ColumnHeaders (contains the ColumnHeader object)
- ColumnHeader
- Format Data (Used for cell formatting)

- DataObjectFiles (Used for drag and drop)
- DataObject (this object)

The DataObject object describes the properties for Drag and Drop. A holding area for formatted text data used in transfer operations. Also holds a list of formats corresponding to the pieces of text stored in the DataObject.

Remarks

A DataObject can contain one piece of text for the Clipboard text format, and one piece of text for each additional text format, such as custom and user-defined formats.

A DataObject is distinct from the Clipboard. A DataObject supports commands that involve the Clipboard and drag-and-drop actions for text. When you start an operation involving the Clipboard (such as GetText) or a drag-and-drop operation, the data involved in that operation is moved to a DataObject.

The DataObject works like the Clipboard. If you copy a text string to a DataObject, the DataObject stores the text string. If you copy a second string of the same format to the DataObject, the DataObject discards the first text string and stores a copy of the second string. It stores one piece of text of a specified format and keeps the text from the most recent operation.

Methods

GetData Method (DataObject Object)

Returns a graphic from the Clipboard object. Doesn't support named arguments.

Syntax

object.**GetData** (format)

The GetData method syntax has these parts:

Part	Description
Object	Required. An object expression that evaluates to an object in the Applies To list.
Format	Optional. A constant or value that specifies the Clipboard graphics format, as described in Settings. Parentheses must enclose the constant or value. If format is 0 or omitted, GetData automatically uses the appropriate format.

Settings

The settings for format are:

Constant	Value	Description
vbCFBitmap	2	Bitmap (.bmp files)
vbCFMetafile	3	metafile (.wmf files)

vbCFDIB	8	Device-independent bitmap (DIB)
vbCFPalette	9	Color palette

Remarks

If no graphic on the Clipboard object matches the expected format, nothing is returned. If only a color palette is present on the Clipboard object, a minimum size (1 x 1) DIB is created.

GetFiles Method (DataObject Object)

Returns a collection of filenames used by a DataObjectFiles collection which in turn contains a list of all filenames used by a DataObject object.

Syntax

object.**GetFiles**(index)

The Files collection syntax has these parts:

Part	Description
Object	An object expression that evaluates to a DataObject object.
index	An integer which is an index to an array of filenames.

Remarks

The Files collection is filled with filenames only when the DataObject object contains data of type vbCFFiles. The DataObject object can contain several different types of data. You can iterate through the collection to retrieve the list of file names.

The Files collection can be filled to allow Visual Basic applications to act as a drag source for a list of files.

GetFormat Method (DataObject Object)

Returns an integer indicating whether an item on the Clipboard object matches a specified format. Doesn't support named argument.

Syntax

object.**GetFormat** (format)

The GetFormat method syntax has these parts:

Part	Description

Object	Required. An object expression that evaluates to a DataObject object.
Format	Required. A value or constant that specifies the Clipboard object format, as described in Settings below. (Parentheses must enclose the constant or value.)

The settings for format are:

Constant	Value	Description
vbCFLink	&HBF00	Text (.txt files)
vbCFText	1	Bitmap (.bmp files)
vbCFBitmap	2	Metafile (.wmf files)
vbCFMetafile	3	Enhanced metafile (.emf files)
vbCFDIB	8	Device-independent bitmap (DIB)
vbCFPalette	9	Color palette

Remarks

The GetFormat method returns True if an item on the Clipboard object matches the specified format. Otherwise, it returns False.

For vbCFDIB and vbCFBitmap formats, whatever color palette is on the Clipboard is used when the graphic is displayed.

Clear Method (DataObject Object)

Removes all objects from a DataObject.

Syntax

object.**Clear**

The Clear method syntax has these parts:

Part	Description
Object	Required. A valid DataObject.

Remarks

The Clear method removes all entries in the list.

SetData Method (DataObject Object)

Inserts data into a DataObject object using the specified data format.

Syntax

object.**SetData** [data], [format]

The SetData method syntax has these parts:

Part	Description
Object	Required. An object expression that evaluates to a DataObject object.
data	Optional. A variant containing the data to be passed to the DataObject object.
format	Optional. A constant or value that specifies the format of the data being passed, as described in Settings.

Settings

The settings for format are:

Constant	Value	Description
vbCFText	1	Text (.txt files)
vbCFBitmap	2	Bitmap (.bmp files)
vbCFMetafile	3	Metafile (.wmf files)
vbCFEMetafile	14	Enhanced metafile (.emf files)
vbCFDIB	8	Device-independent bitmap (DIB)
vbCFPalette	9	Color palette
vbCFFiles	15	List of files
vbCFRTF	-16639	Rich text format (.rtf files)

Remarks

The data argument is optional. This allows you to set several different formats that the source component can support without having to load the data separately for each format. Multiple formats are set by calling SetData several times, each time using a different format. If you wish to start fresh, use the Clear method to clear all data and format information from the DataObject.

The format argument is also optional, but either the data or format argument must be specified. If data is specified, but not format, then Visual Basic will try to determine the format of the data. If it is unsuccessful, then an error is generated. When the target requests the data, and a format was specified, but no data was provided, the source's OLESetData event occurs, and the source can then provide the requested data type.

It's possible for the GetData and SetData methods to use data formats other than those listed in Settings, including user-defined formats registered with Windows via the

RegisterClipboardFormat() API function. However, there are a few caveats:

- The SetData method requires the data to be in the form of a byte array when it does not recognize the data format specified.
- The GetData method always returns data in a byte array when it is in a format that it doesn't recognize, although Visual Basic can transparently convert this returned byte array into other data types, such as strings.
- The byte array returned by GetData will be larger than the actual data when running on some operating systems, with arbitrary bytes at the end of the array. The reason for this is that Visual Basic does not know the data's format, and knows only the amount of memory that the operating system has allocated for the data. This allocation of memory is often larger than is actually required for the data. Therefore, there may be extraneous bytes near the end of the allocated memory segment. As a result, you must use appropriate functions to interpret the returned data in a meaningful way (such as truncating a string at a particular length with the Left function if the data is in a text format).

Editing Control Objects

InaGrid Editing Controls (ColumnHeaders Object)

The InaGrid control ships with several editing controls that can be used to capture user input into an InaGrid cell. The grid control is designed to recognize these editing controls and use them with very little intervention by the application.

All editing controls support the standard OCX property hWnd. hWnd provides a general way to access and initialize these objects. Although this property is not required to use the editing controls with the InaGrid control, it can be useful when the container application needs to access the control. The standard OCX property *Caption* is used by the InaGrid control object to initialize and extract text data from the controls. The standard OCX event *KeyDown* is defined in the editing controls. It is handled by the InaGrid control to respond when the user ends the editing process by pressing the RETURN, ESC or TAB key. And finally a custom event, *OnNoFocus* (ID = 2000) is defined in the controls. It is called when an editing control loses focus. The InaGrid control uses this event to process the user's edit. This usage is the only requirement for custom OCX controls that can be created in the InaGrid control object and edited in the OnEditCell event.

Normally the InaGrid control creates the instance of the editing control and passes a pointer to the instance in the OnInitEditCell event. By trapping this event, the application is able to use this pointer to access properties and methods on the editing control object.

The available controls are:

- InaEdit
- InaCombo Control
- InaCheck Control

InaEdit Control Object

InaEdit Control Object (ColumnHeaders Object)

This is one of the three editing controls that can be used to capture user input into an InaGrid cell. The grid control is designed to recognize these editing controls and use them with very little intervention by the application.

By default, the InaEdit control is created by the InaGrid control when the user initiates an editing action. To change this behavior, the application must override the OnEditCell event.

The other editing controls are InaCheck and InaCombo.

Properties

Caption Property (InaEdit Object)

Returns or sets the text displayed in the InaEdit control.

Syntax

VB

Object.**Caption** [= str]

C++

CString *Object*.**GetCaption**()
void *Object*.**SetCaption**(LPCTSTR str)

Part	Description
Object	An object expression that evaluates to an InaEdit control.
str	A string expression specifying the text appearing in the edit control.

Example

VB

```
Dim strOldEditCaption As String
strOldEditCaption = pControl.Caption
PControl.Caption = "InaEdit Control"
```

C++

```
CString strEditCaption = m_inaEdit.GetCaption();
IGridEdit inaEdit(pControl);
CString strData = “InaEdit Control”;
inaEdit.SetCaption(strData);
```

```
inaEdit.DetachDispatch();
```

hWnd Property (InaEdit Object)

Returns a handle to the InaEdit control.

Syntax

VB

Object.**hWnd**

C++

OLE_HANDLE *Object.***GetHWnd**()

Part	Description
Object	An object expression that evaluates to an InaEdit control.
hWnd	A handle to the InaEdit control window.

Remarks

The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

Trap the *OnInitCellEdit* event to gain access to the editing control in order to access its properties and methods.

This is read-only property.

Note: Because the value of this property can change while a program is running, never store the hWnd value in a variable.

Example

VB

```
Dim editHwnd As OLE_HANDLE, lResult As Long

editHwnd = pControl.hWnd
'Select the entire text
lResult = SendMessage(editHwnd, EM_SETSEL, 0, -1)
```

C++

```
HWND hWndControl = m_editControl.GetHWnd();
IGridEdit inaEdit(pControl);
CString strData = "InaEdit Control";
inaEdit.SetCaption(strData);
::SendMessage((HWND)inaEdit.GetHWnd(), EM_SETSEL, (WPARAM)strData.GetLength(),
(LPARAM)strData.GetLength());
inaEdit.DetachDispatch();
```

InaCombo Control Object

InaCombo Control Object

See Also FAQ Properties Methods Events Enums

This is one of the three editing controls that can be used to capture user input into an InaGrid cell. The grid control is designed to recognize these editing controls and use them with very little intervention by the application.

The other editing controls are InaEdit and InaCheck.

Properties

Caption Property (InaCombo Object)

See Also FAQ Properties Methods Events Enums

Returns or sets the text displayed in the InaCombo control.

Syntax

VB

Object.**Caption** [= str]

C++

CString *Object*.**GetCaption**()
void *Object*.**SetCaption**(LPCTSTR str)

Part	Description
Object	An object expression that evaluates to an InaCombo control.
str	A string expression specifying the text appearing in the InaCombo control.

Example

VB

```
Dim strOldComboCaption As String
strOldComboCaption = pControl.Caption
pControl.Caption = “InaCombo Control”
```

C++

```
CString strComboCaption = m_inaCombo.GetCaption();
_DInaCombo inaCombo(pControl);
inaCombo.SetCaption(“InaCombo Control”);
inaCombo.DetachDispatch();
```

hWnd Property (InaCombo Object)

Returns a handle to the InaCombo control.

Syntax

VB

Object.**hWnd**

C++

OLE_HANDLE *Object*.**GetHWnd**()

Part	Description
Object	An object expression that evaluates to an InaCombo control.
hWnd	A handle to the InaCombo control window.

Remarks

The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

Trap the OnInitCellEdit event to gain access to the editing control in order to access its properties and methods.

This is read-only property.

Note: Because the value of this property can change while a program is running, never store the hWnd value in a variable.

Example

VB

```
Dim editHwnd As OLE_HANDLE

comboHwnd As OLE_HANDLE, lResult As Long
comboHwnd = pControl.hWnd
lResult = SendMessage(comboHwnd, CB_ADDSTRING, 0, “Choice 1”)
lResult = SendMessage(comboHwnd, CB_ADDSTRING, 0, “Choice 2”)
lResult = SendMessage(comboHwnd, CB_SETCURSEL, 1, 0)
```

C++

```
HWND hWndControl = m_editControl.GetHWnd();
_DInaCombo inaCombo(pControl);
    ::SendMessage((HWND)inaCombo.GetHWnd(), CB_ADDSTRING, NULL, (LPARAM)”Combo
Item1”);
    ::SendMessage((HWND)inaCombo.GetHWnd(), CB_ADDSTRING, NULL, (LPARAM)”Combo
Item2”);
    ::SendMessage((HWND)inaCombo.GetHWnd(), CB_ADDSTRING, NULL, (LPARAM)”Combo
Item3”);
```

```
    ::SendMessage((HWND)inaCombo.GetHWnd(), CB_SETCURSEL, (WPARAM)nSel, NULL);
if(-1 ==
    nSel)
  inaCombo.SetCaption("No selection");
    inaCombo.DetachDispatch();
```

InaCheck Control Object

InaCheck Control Object

See Also FAQ Properties Methods Events Enums

This is one of the three editing controls that can be used to capture user input into an InaGrid cell. The grid control is designed to recognize these editing controls and use them with very little intervention by the application.

The other editing controls are InaEdit and InaCombo.

Properties

Caption Property (InaCheck Object)

See Also FAQ Properties Methods Events Enums

Returns or sets the text displayed in the InaCheck control.

Syntax

VB

Object.**Caption** [= str]

C++

CString *Object*.**GetCaption**()
void *Object*.**SetCaption**(LPCTSTR str)

Part	Description
Object	An object expression that evaluates to an InaCheck control.
str	A string expression specifying the text appearing in the InaCheck control.

Example

VB

```
Dim strOldCheckCaption As String
pControl.Init "Check", "Yes", "No"
'Set checked state
pControl.Caption = "Yes"
```

C++

```
CString strCheckCaption = m_inaCheck.GetCaption();
_DInaCheck inaCheck(pControl);
inaCheck.Init("Check", "Yes", "No");
inaCheck.SetCaption("Yes"); ////Set checked state
inaCheck.DetachDispatch();
```

Remarks

Setting the caption changes the check state of the check box. If the caption text is the ON text--the check box is checked; if the caption text is the OFF text--the check box is unchecked. If the caption text is anything else, the check box is placed in the third (intermediate) state. For ON and OFF texts, see *Init* method.

hWnd Property (InaCheck Object)

Returns a handle to the InaCheck control.

Syntax

VB

Object.**hWnd**

C++

OLE_HANDLE *Object.***GetHWnd**()

Part	Description
Object	An object expression that evaluates to an InaCheck control.
hWnd	A handle to the InaCheck control window.

Remarks

The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

Trap the *OnInitCellEdit* event to gain access to the editing control in order to access its properties and methods.

This is read-only property.

Note: Because the value of this property can change while a program is running, never store the hWnd value in a variable.

Example

VB

```
Dim checkHwnd As OLE_HANDLE, lResult As Long
checkHwnd = pControl.hWnd
```

```
lResult = SendMessage(checkHwnd, BM_SETSTYLE, BS_3STATE, 0)
lResult = SendMessage(checkHwnd, BM_SETCHECK, BST_INDETERMINATE, 0)
```

C++

```
HWND hWndControl = m_editControl.GetHWnd();
_DInaCheck inaCheck(pControl);
inaCheck.Init("Check", "Yes", "No");
::SendMessage((HWND)inaCheck.GetHWnd(), BM_SETSTYLE, (WPARAM)(BS_3STATE),
NULL);
::SendMessage((HWND)inaCheck.GetHWnd(), BM_SETCHECK, (WPARAM)
(BST_INDETERMINATE),
NULL); //Set grayed state
inaCheck.DetachDispatch();
```

Methods

Init Method (InaCheck Object)

Initializes the title of the check box and tests for checked and unchecked state as they will be accessed by Caption property.

Syntax

VB

object.Init pszTitle, pszOn, pszOff

C++

void Object.Init(LPCTSTR pszTitle, LPCTSTR pszOn, LPCTSTR pszOff)

The Init method syntax has these parts:

Part	Description
Object	An object expression hat evaluates to a ColumnHeaders collection object.
pszTitle	The title of the check box.
pszOn	The string corresponding to checked state of the check box.
pszOff	The string corresponding to unchecked state of the check box.

Example

VB

```
pControl.Init "Check", "Yes", "No"
lResult = SendMessage(pControl.hWnd, BM_SETSTYLE, BS_3STATE, 0)
lResult = SendMessage(pControl.hWnd, BM_SETCHECK, BST_INDETERMINATE, 0)
```

C++

```
_DInaCheck inaCheck(pControl);
inaCheck.Init("Check", "Yes", "No");
::SendMessage((HWND)inaCheck.GetHWnd(), BM_SETSTYLE, (WPARAM)(BS_3STATE),
```

```
NULL);
::SendMessage((HWND)inaCheck.GetHWnd(), BM_SETCHECK, (WPARAM)
(BST_INDETERMINATE), NULL); //Set grayed state
inaCheck.DetachDispatch();
```

constant

A named item that retains a constant value throughout the execution of a program, as opposed to a variable, whose value can change during execution. Each host application can define its own set of constants. Additional constants may be defined by the user with the Const statement. Constants can be used anywhere in your code in place of actual values. A constant may be a string or numeric literal, another constant, or any combination that includes arithmetic or logical operators except Is and exponentiation. For example:

Const A = "MyString"

object library

A file with the .OLB extension that provides information to Automation controllers (like Visual Basic) about available Automation objects. You can use the Object Browser to examine the contents of an object library to get information about the objects provided.

Object Browser

You can use the Object Browser to examine the contents of an object library to get information about the objects provided.

object expression

An expression that specifies a particular object. This expression can include any of the object's containers. For example, your application can contain an Application object that contains a Document object that contains a Text object.

Cascading event

A sequence of events caused by an event procedure directly or indirectly calling itself; also referred to as an event cascade or recursion. Cascading event procedures often result in run-time errors, such as stack overflow.

access key

A key pressed while holding down the ALT key that allows the user to open a menu, carry out a command, select an object, or move to an object. For example, ALT+F opens the File menu.

focus

In the Microsoft Windows environment, only one window, form, or control can receive mouse clicks or keyboard input at any one time. The object that "has the focus" is usually indicated by a highlighted caption or title bar. The focus can be set by the user or by the application.

control array

A group of controls that share a common name, type, and event procedures. Each control in the array has a unique index number that can be used to determine which control recognizes an event.

bit masks

A value used with bit-wise operators (And, Eqv, Imp, Not, Or, Xor) to test, set, or reset the state of individual bits in a bit-field value.

function keys

Any of the keys labeled F1 through F12. Function keys often provide shortcuts for frequently carried out commands and actions. You can assign a function key as a shortcut key.

editing keys

The INSERT, DELETE, or BACKSPACE key.

ANSI character set

American National Standards Institute (ANSI) 8-bit character set used by Microsoft Windows that allows you to represent up to 256 characters using your keyboard. The first 128 characters correspond to the letters and symbols on a standard U.S. keyboard. The second 128 characters represent special characters, such as letters in international alphabets, accents, currency symbols, and fractions.

G

H

I