TrueGrid

A Data-Aware Grid for Visual Basic 3.0 Data Access

This is a demo version of TrueGrid.   We believe we've created the easiest-to-use database grid available and appreciate your interest in this demo.   It is identical in features and functionality to the full TrueGrid product, except that it will only operate in Visual Basic design mode.   We want to make it easy for you to get a good idea of what TrueGrid offers, how it operates, and how flexible it is.

We've put together a demo program which runs during design-time.   The demo program is called **TGDEMO.MAK.**   To run the demo, you should:

1.   Run Visual Basic.

2.   Open the TGDEMO.MAK project.

3.   Run the application.

The TGDEMO program is self-explanatory and has a window called the Navigator which describes each of the demos and how to operate them.

Running TGDEMO is the quickest way to understand how TrueGrid operates, but you can also develop your own design-time applications using the demo.   We've provided thorough documentation with the demo, so you can understand how all of the properties and events operate.

If you like TrueGrid, and want one for your very own, you can order one directly from Apex at 1-800-858-APEX.   You can also fax in your order at 1-412-681-4384.   For other questions and information, call Apex at 1-412-681-4343.   We would be glad to provide you with some assistance with the demo, but can't provide the level of support we offer for product owners.   The easiest way to contact us is by using CompuServe.   Brian Hess (71053,1062) would be glad to help you and answer any of your questions.

As of this writing (August 1, 1993), TrueGrid is $69 when purchased from Apex.

This help file is a trimmed-down version of the actual help file which ships with TrueGrid.   To save space, we've omitted the sections describing the examples which ship with TrueGrid (you don't have those anyway).   We've also removed some of the larger graphic images to save space.

If you want, you can begin developing using the demo version of TrueGrid, since it has the same product features as the production version.   However, when you purchase a production TrueGrid, you will need to recreate any TrueGrid controls you were using in the demo and reset the design-time properties they way they were originally.

Here's the table of contents for the on-line TrueGrid documentation:
- Overview of TrueGrid
- How to use TrueGrid's features, step-by-step
- Summary of Features
- Property Reference
- Event Reference

How to Use TrueGrid

Configuring TrueGrid Database Properties

Associating a grid with a Visual Basic Data Control
How field and column layouts are chosen

Working with TrueGrid Columns

Design time column layouts using the Layout Editor
Adding columns at run time
Adjusting column properties using Visual Basic code
Allowing the user to reconfigure grid appearance at runtime
Using column edit masks

TrueGrid Database Application Techniques

Using TrueGrid for SQL query results
Data validation techniques
Coordinating multiple TrueGrid controls
Using the BookmarkList to highlight query result rows

How Your Programs Interact with TrueGrid

How TrueGrid reacts to Dynaset changes
Data Control interaction and how TrueGrid responds
TrueGrid live data updates and Validate handlers
Row numbers and what they mean
Eliminating unwanted grid refresh

Configuring the Color and Appearance of TrueGrid

TrueGrid color settings
How to give the grid a 3-dimensional appearance
Displaying grid lines between rows or columns
Changing the appearance of the marquee

Using TrueGrid's Multiple Selection Feature

Multiple selection overview
How the user selects multiple rows
Working with the list of marked rows in your programs
Clearing marked rows using Visual Basic code

Using TrueGrid in Callback (Application) mode

Introduction to TrueGrid Callback mode
Steps to create a Callback mode application
Creating a user friendly callback program

◩ Overview

TrueGrid simplies programming by being an easy to use data bound grid for Visual Basic 3.0.   The technology is derived from the data grid we provide with our Agility database product, so it has been well tested and thouroughly enhanced to satisfy the peculiarities of database usage.

The major attraction is that the grid is very easy to use, requiring no more skill than simply dropping the grid on the form and setting the DataSource property---that's it!   You don't need to write a single line of code to get the grid up and running.   It's ideal for a data browser or display window.

The grid works with all formats that Visual Basic recognizes.   Any database that you can use with Visual Basic you can use with the grid.

We've spent a lot of time trying to make TrueGrid *the* inexpensive alternative for Visual Basic data access users.   We hope you enjoy using it.   If you need technical assistance, please call and talk to us.   We'd be glad to help.

The How to Use TrueGrid section provides detailed examples and instructions on how to use the grid.   If you're already very familiar with Visual Basic and other custom controls, you may want to quickly peruse the Property Listing and Event Listing

◾ Features

The following is a highlight of some of the major features of TrueGrid:

1. Drop-and-go behavior.   Drop the grid on the form and setting <u>DataSource</u> are all you need to do to have a working, editable, data grid.

2. Responds to changes you make to the Data control or Dynaset using a deferred response scheme.   So, if your program does a MoveNext 10 times, the grid will respond only once by moving the highlighted row.   This makes programming effortless, since the grid worries about efficiency and all you worry about is you program.

3. Updatable.   Records can be changed in the grid and are updated automatically to the database.

4. Handles databases of any size.   No memory overhead for large tables.

5. Responds to database movement, queries, FindFirst, Refresh, UpdateData, UpdateControls, RecordSet changes via SQL, etc.   In other words, everything you do will be reflected in the grid.

6. Customizable color schemes, 3D appearance, row highlighting styles.

7. Automatic per-column edit masks using Format$ strings.

8. Column layout can be set up at design time, using the Layout Editor, or will automatically be derived at runtime.

9. Column properties for fieldname, heading, data size, column width, editability, justification, edit mask.

10. Per-column validation routines.

11. Works with all database formats supported by VB3.

Configuring TrueGrid Database Properties

■ Associating a grid with a Visual Basic Data Control
■ How field and column layouts are chosen

Associating a grid with a Visual Basic Data Control

You can make an association between a TrueGrid data bound control and a Visual Basic data control by setting the <u>DataSource</u> property of the grid after the data control has been created on the same form. This is all you need to do to make TrueGrid fully aware of the database in your application.

Once such a link exists, TrueGrid respects the information present in the data control to simplify further refinement of your application:

> Once the grid has been associated with a data control, you can click on the <u>Layout</u> property and TrueGrid will ask whether you want to use the field layout of the database automatically.

> When you run you application, TrueGrid will automatically respond by displaying the contents of the database opened by the data control.   Data will be fully available at runtime, and can be edited as well.

Any changes you make to the data control will be automatically sensed by TrueGrid and it will update its display accordingly.   However, TrueGrid waits until the system is idle before performing such display updates, since programs may want perform many actions before it is necessary to syncrhonize the display.

How field and column layouts are chosen

In a TrueGrid display, each column represents a single field of data.   For each column, the grid needs to know the *fieldname* associated with the data, and optionally a heading to be displayed above the data column.

TrueGrid gets information about field names and headings in one of three ways:

1. At design time, you can select the Layout editor to specify column information manually.

2. At design time, when you attempt to use the Layout editor for the first time, TrueGrid asks if you want to use the field definitions in the database rather than definining them yourself.   This makes the initial layout simple to create, and you can easily modify it with further use of the layout editor.

3. If you don't define a layout at design time at all, TrueGrid will automatically create one based upon the database used when you run your program.   This mode is the most automatic of all and is quite acceptable for most quick applications.

At run time, you can perform database actions which may alter the layout needed to display the data. For example:

1. You may change the Database or Connect strings in the Data Control, which results in a different database.

2. You may change the RecordSource property of the Data Control (for example, to invoke an SQL query).

In all such cases, the grid will automatically sense the new column layout and reconfigure itself.   But, if you defined a design-time layout yourself, the grid won't change the layout at all, assuming that you want total control of the grid's appearance.

Working with TrueGrid Columns

■ Design time column layouts using the Layout Editor
■ Adding columns at run time
■ Adjusting column properties using Visual Basic code
■ Allowing the user to reconfigure grid appearance at runtime
■ Using column edit masks

Design Time Column Layouts using the Layout Editor

The Layout Editor can be used at design time by clicking on the Layout property.   Alternately, you can click with the right mouse button over a specific grid to make the layout editor appear.   Each grid has its own layout of columns, and the layout editor can be used to add, remove, and modify the column layout. All of the layout information is saved as part of your Visual Basic project, so it will be used at runtime automatically.

If your grid is associated with a database, then you will be prompted to find out if you want to use the field layout of the database for your initial column layout.   If you say No, then you can design your column layout from scratch anyway you wish.

The layout editor shows a copy of the grid control in a dialog, with a special menu which can be popped up using the right mouse button:

    [Picture of Layout Editor omitted to save space for demo help file.]

The pop-up menu controls settings for only a single column---the column you clicked over.   For the Heading Text, Field Name, and Edit Mask properties, choosing an item allows you to edit the respective text.   All text editing occurs inside the heading area for the given column, although the text normally displayed there is the Heading Text item.

Adding Columns at Run Time

Using grid properties and Visual Basic code, you can add columns to the grid at runtime.   You can even perform complete grid configuration yourself, rather than using the interactive tools.

Here is an example of how a column can be added to the grid at runtime:

```
nextc% = Table1.Columns + 1 ' index of next column

' add a column with new properties
Table1.ColumnField(nextc%) = "firstname"
Table1.ColumnName(nextc%) = "First Name"
Table1.ColumnWidth(nextc%) = 25
```

Note that simply assigning to one of the column properties with a column number one greater than the final column will automatically add a new column.   (Note that just after the first assignment, Table1.Columns returns the new, higher value).

Once added, the new column will be automatically linked to the DataSource database, if one is opened.

You cannot remove columns invidually, but you can hide them (using the ColumnStyle) property.   You can also remove all of the columns from the grid layout with the following line of code:

```
Table1.Columns = 0
```

Adjusting Column Properties using Visual Basic Code

Existing column properties can be changed at runtime using Visual Basic code.   For example, changing ColumnField can be done as follows:

```
Table1.ColumnField(3) = "DateofBirth"
```

When this statement is executed, the grid will automatically refresh any data in column 3 by refreshing it from the database.   Thus, such changes will be reflected automatically by the grid.

Here are the column properties you may wish to change:

ColumnName	The heading for the specified column.
ColumnField	The field name associated with the specified column.
ColumnWidth	The display width, in characters of the grid's font.
ColumnSize	The maximum data entry width, in characters.
ColumnStyle	The column justification and visibility bits.
EditMask	The edit mask for the column.

Changing any of the above properties causes the grid to respond by updating its display to reflect the change.

Allowing the User to Reconfigure Grid Appearance at Runtime

If the Configurable is *True* (which is the default), then TrueGrid allows the user to interactively move and adjust the column layout at runtime.

All user adjustment is made with the mouse, as follows:

1. If VertLines is set so that vertical column separators are visible, then a special "move line" cursor will appear when the user has the mouse positioned over a vertical line.   In such cases, the user may grab the separator and drag it to the left or right, thereby enlarging or shrinking the column immediately to the left of the line.

2. If Headings is *true*, making grid column headings visible, then the user may hold down the mouse over a column heading.   The entire column becomes highlighted and it can be dragged to the left or right to change the column ordering.   Note that this *does not* change the indexes used in your program to refer to columns in the grid.   No matter where the user moves the leftmost column, it will remain column 0 to your program.   (This is accomplished by having a separate property, ColumnOrder, which designates the visual order of columns.)

3. Columns may be shrunk to nothing (using the technique in item 1), in which case the ColumnVisible property for the column is set to *false*, although the column is not physically removed from the grid.

If the grid is not configurable (the Configurable property is set to *false*), then no user configuration will be permitted and all mouse events are passed to standard event handlers such as MouseDown.

Using Column Edit Masks

Each column has an optional edit mask associated with it.   The edit mask can be set in two ways:

1. The EditMask menu item can be selected from the Layout Editor at design time to set an edit mask for the column.

2. At run time, the EditMask property can be set in Visual Basic code to specify an edit mask for the column.   The column will automatically be refreshed to reflect the newly edited appearance of the columns.

Edit masks in TrueGrid are the standard Visual Basic Format Strings used in the Format$ function.

The edit mask is used only for the *display* of data in the grid.   When a cell is opened for editing, the actual underlying data is displayed, rather than the edited value.   When editing is complete, TrueGrid displays the changed data in red until database motion or an Update method causes the data to be written back to the database.   This tells the user that the data has been changed, but still hasn't been updated.   The grid displays the changed data *without* applying the EditMask so that the actual data can be examined.   Once the data is updated, TrueGrid applies the edit mask to the data and removes the red highlighting.

TrueGrid Database Application Techniques

- Using TrueGrid for SQL query results
- Data validation techniques
- Coordinating multiple TrueGrid controls
- Using the BookmarkList to highlight query result rows

Using TrueGrid for SQL Query Results

One important use for TrueGrid's ability to automatically sense changes to the database at runtime is for displaying the results of ad-hoc SQL queries.

For example, assume that a grid is connected to the data control, Data1, and that a query button is provided which contains code similar to this:

```
Sub QueryButton_Click ()
    ' Copy SQL query to RecordSource
    Data1.RecordSource = QueryText.Text

    ' Refresh data set ... connected grid will respond
    ' automatically

    Data1.Refresh
End Sub
```

This simple example illustrates two important points:

1. No code is necessary to tell the grid what to do.   The change to the RecordSource property will be detected by the grid and it will display the new query result automatically.

2. The grid will automatically change its field layout to match the new configuration of the query result.   This can be important if the query uses aggregate functions (such as **count()**) or specifies alternate field lists or calculated fields.

The second point is especially interesting.   Imagine an SQL query such as this:

```
SELECT COUNT(*) FROM EMP
```

The above query returns a single row, containing a single value which is the number of rows in the database EMP.   The grid will display the result like this:

[Picture of COUNT result omitted to save space for demo help file.]

Notice that a single column is displayed (called "Calc#0" since no name was given to the result field) and a single query row containing the count result (641) is displayed.

Similarly, you can select only some fields in a query, such as the following:

```
SELECT FIRST, LAST, TITLE FROM EMP WHERE ZIP >= 90000
```

The above query will select only three fields, but contingent upon a zip code condition.   The result looks like this:

[Picture of query result omitted to save space for demo help file.]

Notice how the field headings have been reconfigured automatically to reflect the new column layout. The grid will also respond to queries which have expressions instead of simple field names:

```
SELECT FIRST, UCASE(LAST), TITLE FROM EMP WHERE ZIP >= 90000
```

Now, the query result includes a calculated column instead, which reflects the result of changing the last name field to upper case:

[Picture of query result omitted to save space for demo help file.]

You can also give calculated columns names by including an "AS" clause in your SQL statement:

```
SELECT FIRST, UCASE(LAST) AS UPPERLAST
          FROM EMP WHERE ZIP >= 90000
```

In this case, the query result table will include the name of the calculated column:

[Picture of query result omitted to save space for demo help file.]

Whether giving users access to SQL statements directly, or using these features in your program, TrueGrid provides a powerful mechanism for direct display of SQL queries.   Since TrueGrid configures itself automatically, very little programming is necessary to create an extremely powerful user interface.

Data Validation Techniques

TrueGrid includes protocols for several data validation techniques.   When data is updated directly using the grid, a special protocol is followed:

1. When a cell is opened for editing, the user can make changes directly to the data in the grid. When completed, the grid <u>Validate</u> event is triggered.   This gives you the immediate opportunity to reject or accept data as it is being entered.   If validation doesn't fail, the grid <u>Update</u> event is triggered, indicating that data has been successfully updated by the user.   *This does not indicate that the data has been written to the database ... only that the user has changed the visible data in the grid.*

    The grid Validate and Update events are useful for providing the user with immediate feedback about the data they have entered.

2. Changed data is held in the grid until it is updated to the database, no matter how many changes are made to the current row.   Data is updated to the database when (1) the record pointer is moved for any reason, (2) when an UpdateRecord method is sent to the data control associated with the grid.

Two grid properties, <u>ColumnChanged</u> and <u>ColumnText</u> are used to determine what data in the current row has been modified by the user.   Of course, you can determine this directly by putting code in the grid Validate and Update handlers, but it is often easier to rely on the grids bookkeeping and check all of the data later.

The ColumnChanged property returns *True* if data in the specified column has been changed by the user, but not yet updated to the database (it appears in red in such circumstances).   The ColumnText property returns the text of the specified column cell.   The text will be the edited text if ColumnChanged is *True* and will be the database text if ColumnChanged is *False* for that column.

For example, in the data control Validate handler, you could examine all of the columns in the grid to determine if all of the changes are valid:

```
' Check all grid data which has been changed

For i% = 1 To Table1.Columns
   If Table1.ColumnChanged(i%) Then
        ValidateData i%, Table1.ColumnText(i%)
   End If
Next i%
```

When data is successfully updated, all of the ColumnChanged values are set to *False* (as is the <u>DataChanged</u> property for the grid, which reflects the overall change state of the grid).

Coordinating Multiple TrueGrid Controls

You can link multiple TrueGrid controls together by using the <u>RowChange</u> event to trigger related actions.

Whenever the highlighted row moves in the grid, the RowChange event is sent, indicating that a new row has become current.   RowChange is triggered in the following cases:

1. If the user clicks on a new row in the grid.

2. If program code moves the record pointer (using methods like MoveFirst, MoveNext, FindNext,

etc.).

3. If the data in the grid is refreshed.
4. If the user moves the record pointer using the navigation buttons on the data control associated with the grid.

The RowChange handler is a convenient place to coordinate activities with other controls or databases.

For example, assume that you have one grid (CustTable) containing customers and another (CustInvTable) containing invoices for the customer.   CustTable is linked to data control CustData and CustInvTable is linked to data control CustInvData.   You could use the RowChange handler in CustTable to perform an SQL selection so that invoices are always displayed as new customers are selected:

```
Sub CustTable_RowChange ()
    ' Get the customer number for the new row
    custid$ = CustData.RecordSet("CustNum")

    ' Build a query for the invoice database
    cquery$ = "SELECT * FROM CUSTINV WHERE CUSTNUM = " & custid$

    ' Invoke the query, which causes the second grid to update
    ' automatically.

    CustInvData.RecordSource = cquery$
    CustInvData.Refresh
End Sub
```

Again, note that no special handling is required for the grids associated with the data controls.   They respond automatically to changes made by such code.

Using the BookmarkList to Highlight Rows

TrueGrid has a powerful multi-row selection feature which relies on bookmarks.   One or more grid rows can be selected (and are displayed differently using the SelectedForeColor and SelectedBackColor).

The grid maintains a list of all selected rows in the BookmarkList property.   Here's how BookmarkList behaves:

1. The user can select a row by moving the cursor to the leftmost area of the grid and clicking. When this happens, the grid automatically adds the newly selected row to the BookmarkList and updates BookmarkCount.   The opposite happens when a user deselects a selected row.   (The SelectMode property has to be set so that this is enabled.)

2. You can examine the contents of BookmarkList at any time to find out which rows are selected.

3. You can set an element of BookmarkList to an empty string ("") to remove the bookmark at that position.   The selected row will no longer be highlighted and BookmarkCount will automatically be decremented.   (TrueGrid will move all the bookmarks up one position to fill the empty slot in the BookmarkList property array.)

4. You can add bookmarks to the bookmark list yourself by adding them at the end.   The grid will automatically highlight any visible rows which correspond to your new bookmarks.

5. By setting BookmarkCount to zero (0), you can clear the entire selection.

This facility is extremely powerful.   For example, you could highlight all rows in the grid containing employees who have the title "Manager" with the following simple subroutine:

```
Sub HighlightManagers()
    ' Set criteria for FindFirst/FindNext
    criteria$ = "TITLE = " + chr$(34) + "Manager" + chr$(34)

    ' Loop through, adding each manager to the BookmarkList
```

```
      Data1.RecordSet.FindFirst criteria$
      While Not Data1.RecordSet.EOF
          Table1.BookmarkList(Table1.BookmarkCount) =
  Data1.RecordSet.Bookmark
          Data1.RecordSet.FindNext criteria$
      Wend
  End Sub
```

When the above loop is complete, all of the bookmarked rows will automatically be highlighted in the grid and the user can view the results.

You can also move the pointer to one of the current bookmarks:

```
  Data1.RecordSet.Bookmark = Table1.BookmarkList(0)
```

This moves the record pointer (and the grid highlighting) to the first selected row.

How Your Programs Interact with TrueGrid

■ How TrueGrid reacts to Dynaset changes
■ Data Control interaction and how TrueGrid responds
■ TrueGrid live data updates and Validate handlers
■ Row numbers and what they mean
■ Eliminating unwanted grid refresh

How TrueGrid Reacts to Dynaset changes

When you link a grid to a data control, you are actually linking the grid to the underlying *Dynaset* which is managed by the data control.   The Dynaset can be accessed directly by using the RecordSet property of the data control:

```
Data1.RecordSet.MoveFirst
```

The above line positions the record pointer of the Dynaset to the first record in the record set.

You needn't worry about keeping the grid synchronized with changes made to the Dynaset.   This happens automatically.   In fact, modifying the Dynaset is the preferred way to affect changes to the grid display.

Here is how the grid responds to various operations performed on the Dynaset associated with the data control:

MoveFirst, MoveNext, MoveLast, MovePrevious
> The grid will move its highlight in response to these methods when they are invoked on the Dynaset.   The grid optimizes its response to these messages.   Thus, if your code performs a MoveFirst followed by 20 MoveNext methods, the grid will position to row 21 only once.   If possible, the grid will adjust its row numbering scheme and scroll bar to reflect the new position within the database.

FindFirst, FindNext, FindLast, FindPrevious
> The grid will move its higlight in response to a new row becoming current after these methods are invoked on the Dynaset.   However, the grid may not know what actual row number the new row possesses.   For example, FindNext may move an unspecified number of rows forward and there is no way to determine how many.   To avoid time-consuming counting operations, the grid simply uses a "fictitious" row number until it gains more information about how many rows precede or follow the new row.   For the user, this behavior is usually undetectable, but results in vast increases in speed over alternative techniques.

AddNew   This method should not be executed without a corresponding Update method to the Dynaset, *unless* the <u>Active</u> property of the grid is *False*.   If the grid is active, and control returns to the grid after an AddNew, the effect of the AddNew will be nullified.   This is because AddNew does not actually add a row, and the grid must have an actual database row to display.   However, if you actually update the new record, the grid will display the new row properly when it receives control.

Update   The update event causes the grid to instantly write any changed data to the database (providing that the data control Validate handler did not suppress the operation).   Any edited cells will be unhighlighted to indicate that the data has now been written.

Refresh   Refreshing the Dynaset causes the grid to redisplay all rows of data, and analyze the field structure of the result set to see if it has changed.   Any changes to headings or field names will be automatically made, unless there is a design-time layout defined using the <u>Layout</u> property.

The data control itself also supports several methods which affect the way the grid reacts to data changes:

UpdateRecord

The data in the grid will be written automatically to the database.

UpdateControls

This method causes any changed data in the grid to be refreshed from the database. Thus, any editing done by the user is lost and the original field data will be displayed once more.

Data Control Interaction and How TrueGrid Responds

The data control panel can be used to navigate through any grids connected to it.   Each of the buttons performs its respective function and the row change is indicated in the grid:

* Pressing the "move to beginning" button has the same effect as a MoveFirst method sent to the Dynaset.   The grid will position to the first row in the set.

* Pressing the "move to last" button has the same effect as a MoveLast method sent to the Dynaset.   The grid will position to the last row in the set.

* The "move to next" and "move to previous" buttons have the same effect as MoveNext and MovePrevious when sent to the Dynaset.   The grid will move the highlight forward or backward by one row.

If you depress the data control buttons rapidly, the grid may have to "catch up" but will eventually resynchronize with the new position.

If grid data has been edited, then moving the pointer using the data control will automatically trigger an update.

TrueGrid Live Data Updates and Validate Handlers

Data in the grid can be updated directly, and validate handlers in both the grid and in the data control can be used to control the outcome of the update.   Some details about how grid editing occurs:

1. When a cell is opened for editing, the user can make changes directly to the data in the grid. When completed, the grid <u>Validate</u> event is triggered.   If validation doesn't fail, the grid <u>Update</u> event is triggered, indicating that data has been successfully modified by the user.

2. Changed data is held in the grid until it is updated to the database, no matter how many changes are made to the current row.   Data is updated to the database when (1) the record pointer is moved for any reason, (2) when an UpdateRecord method is sent to the data control associated with the grid.

Two grid properties, <u>ColumnChanged</u> and <u>ColumnText</u> are used to determine what data in the current row has been modified by the user.   The ColumnChanged property returns *True* if data in the specified column has been changed by the user, but not yet updated to the database (it appears in red in such circumstances).   The ColumnText property returns the text of the specified column cell.

TrueGrid uses bookmarks internally to navigate through the database.   Thus, whenever a data control's Validate handler is called in response to grid movement, the DATA_ACTIONBOOKMARK action will indicate that movement is about to occur.   In your validate handler, you have the following options:

1. You can cancel the movement, by setting the **Action** parameter to 0.   In such a case, the grid will return the highlight to the original row rather than the newly selected row.   If you set the **Save** parameter to *False*, then any user edits will remain and the user will need to change something before continuing.   This is (typically) how invalid data is handled.

2. You can allow the movement to continue, but ignore any changes made by the grid.   You can do this by setting the grids <u>DataChanged</u> property to *False*, which causes the grid to eliminate any user edits it is currently displaying.

Avoid setting the **Action** parameter to any other value, since the grid behavior may not be as you expect (although the grid will properly handle such changes).   For example, if you inadvertently set **Action** to DATA_ACTIONMOVEFIRST, then any attempt to change to another row will force the grid back to the first row.

Row Numbers and What They Mean

TrueGrid uses *row numbers* to identify rows in the grid, but in general, these row numbers are meaningless except for some programming purposes.   This section explains why this is so.

First, Visual Basic databases and sets do not have row numbers at all.   You can move to a particular row by moving to the first or last record, then counting from there.   If you are at a particular row, it can be quite time consuming to determine what row you are at.   For example, if you are at an unknown row, you could remember its bookmark, then go to the first row, counting forward until you reach the bookmarked row.

This works well for very small databases, but for larger databases, such determinations can take minutes (or even hours).   This is a limitation (or rather a feature, actually) of Visual Basic's data access system and cannot be avoided.

TrueGrid was designed with *efficiency* as its primary concern.   Thus, although knowing exactly what row you are at can be convenient for the programmer, it is disasterous in terms of overhead.   Thus, the grid operates in one of three different modes, and it does so transparently to the user:

1. *Neither the first nor last row number is known.*   This situation occurs more often than you think. For example, imagine that a FindNext command has moved to a record *somewhere* in a 400000 record database.   There is no way to determine whether you are at row 3000 or 300000, and finding out could take 20 minutes of searching.   To avoid this, the grid simply assumes that there may be an unknown number of rows before and after the new row.   In such cases it makes up a *fictitious* row number (usually 500000) and displays records near the current record.   The display thus occurs instantly, and as you move forward or backward, the grid moves relative to the fictious position until it actually hits the beginning or end of the set.   In the latter case, it updates its information so that either the first or last row number are now known, and starts using real row numbers.

   All of this is transparent and leads to very rapid operation.   One side effect is that the scroll bars are sometimes approximate.   In the above case, the scroll bar would be positioned roughly in the middle, since the grid doesn't know where in the database you are actually positioned.

2. *The last row number is unknown.*   This is the usual cirumstance when browsing a new set which is too large to display.   Rather than try to determine how many rows there are in the set, TrueGrid assumes that there may be an infinite number of rows, and displays only those necessary.   As you move forward, the grid will detect the last row when encountered and update its internal information about the size of the set.

   In this case, the scroll bar moves more slowly as you move further down in the database, since the grid doesn't really know how many records there may be.   Once the total number of rows is known, the scroll bar will behave in a linear fashion.

3. *First and last row number are known.*   In this case, the grid's Rows and RowIndex property accurately reflect the actual row within the set.   The grid's scrollbar will be very accurate as well. The grid knows you are at the first row when you do a MoveFirst, and knows the last row when you do a MoveLast.   Both of these operations (as well as scrolling to the first or last record) will cause the grid to sense the actual row position of the data.

Once you are aware of these factors, you can more readily understand and use the values of the Rows and RowIndex properties.

RowIndex can always be used to position to the first row:

```
Table1.RowIndex = 1
```

If the grid doesn't know what the first row is, it will determine the row number in order to satisfy your request.   Similarly, you can always position to the last row:

```
Table1.RowIndex = Table1.Rows
```

Again, the grid will determine the total number of rows in order to position to the last row properly.   You

can also cause such positioning by sending the **MoveFirst** or **MoveLast** method to the corresponding Dynaset associated with the grid.

So long as the grid does change from one "mode" to another, you can use the RowIndex to move the highlight relative to visible locations in the grid.   For example, you can move the highlight to the first row visible in the grid:

```
Table1.RowIndex = Table1.TopRow
```

or the last on the screen:

```
Table1.RowIndex = Table1.BottomRow
```

You can move the highlight forward or backward:

```
Table1.RowIndex = Table1.RowIndex + 1
Table1.RowIndex = Table1.RowIndex - 1
```

Setting RowIndex is *not* the recommended way to move the record pointer, although it will work.   It is best to manipulate the Dynaset directly using methods such as MoveFirst, MoveNext.   Setting RowIndex is useful only for special cases.

Eliminating Unwanted Grid Refresh

Normally, TrueGrid is very intellegent about deferring screen updates until it absolutely necessary.   You should rarely need to worry about the grid's operation, and simply write code which works directly with the database.

How does the grid sense that it may refresh?   It waits until you enter the Windows *idle loop*.   This generally occurs when your code stops executing and the system is waiting for user input.   You can *force* the grid to think the idle loop has been entered by executing a Visual Basic **DoEvents** call, which causes all controls to become active.

In some circumstances, you may want the grid to remain "frozen" as it is and not respond to any database changes you make.   To do this, set the grid's Active property to *False* temporarily while you perform whatever action is necessary, then set it to *True* again when you want it to resynchronize with the database.   Although you should rarely have to do this, one situation which demands this is the use of a modal form which pops up while the database is in an uncertain state.   If you don't set Active to *False*, then the grid will update to reflect the new database state.   If you want to avoid this while processing a modal dialog, deactivate the grid when your dialog is displayed, and reactivate it when the dialog is complete.

It is dangerous to leave the grid Active property set to *False* for any longer than necessary.   When the Active property is *False*, the database position and the grid position are permitted to diverge.   This can cause confusion for the user.   For example, the highlighted row may not be the current row at all, and attempts to edit the data in the grid may edit the wrong row.

Configuring the Color and Appearance of TrueGrid

■ TrueGrid color settings
■ How to give the grid a 3-dimensional appearance
■ Displaying grid lines between rows or columns
■ Changing the appearance of the marquee

TrueGrid Color Settings

TrueGrid supports a wide variety of color settings to assist you in customizing the appearance of the grid.

Here are some of the colors you can change, and why you might want to change them:

ForeColor, BackColor

>These colors specify the foreground and background color for the entire grid and normal displayed text.   For 3D effects, BackColor must be set to light grey.

EditBackColor, EditForeColor

>When a cell is opened for editing, it is changed to this color.   By default, the system highlight colors are used, but this can make it difficult to see the text selection of the edited text.

HeadBackColor, HeadForeColor

>These colors are used for the heading area of the grid (providing that the Headings property is *True*).   These colors are used only when the grid has focus.   When the grid doesn't have focus, the InactiveForeColor and InactiveBackColor are used for the headings.

InactiveBackColor, InactiveForeColor

>These colors are used for the heading area of the grid when it doesn't have focus.

SelectedBackColor, SelectedForeColor

>These colors are used for the display of rows which have been selected by the user, or which are selected because their bookmarks have been added to the BookmarkList property.

VertColor, HorzColor

>These colors are used in conjunction with the HorzLines and VertLines properties to specify the color of the grid separator lines.   For a 3D appearance, these colors should be set to dark grey and the HorzLines and VertLines property should be set for 3D line appearance.

How to Give the Grid a 3-Dimensional Appearance

TrueGrid supports both a standard, "flat" control appearance and the more attractive 3D appearance used by many controls.   By default, the grid appears as a standard, non-3D control.   However, you can achieve a 3D appearance by setting the following properties at design-time:

1.   Set both the HorzLines and VertLines properties to "3D" (integer value=2).

2.   Set the BackColor property to light grey.

3.   Set the HorzColor and VertColor properties to dark grey.

That's all there is to it.   The grid will display cells using a 3D appearance, and the marquee will display a raised current cell when it is set to value 4 (Highlight Row and Cell).

You can achieve a 3D effect using other color combinations, but you will need to experiment.   For example, a rather offensive 3D display can be concocted using light and dark yellow instead of grey in the above examples.   In our experience, grey remains the best.

Displaying Grid Lines Between Rows or Columns

The HorzLines and VertLines properties allow you to choose whether horizontal and vertical lines are used in the grid.   Both are optional, and each have three values:

0 - None.   No lines are displayed in the given direction.

1 - Single.   A single-width line is displayed using the color specified by HorzColor or VertColor.

2 - 3D.   3-dimensional lines are used.   In this case, HorzColor and VertColor specify the darker edge of the 3D bevel and the grid interpolates to find a lighter color for the opposite size of the bevel.

Lines cause slightly less data to be displayed in the grid, since they take up space.   You may want to experiment with different settings to find out what is appealing to you.

Changing the Appearance of the Marquee

The *marquee* is the name of the higlighted area which represents the current cell or row of the grid.   The MarqueeStyle property can be set to have 5 possible presentations, illustrated below.

MarqueeStyle = 0 - Dotted Cell Border

This is the default appearance of the marquee:

MarqueeStyle = 1 - Solid Cell Border

This is a more distinctive form of cell highlighting, often useful when a different background color is used (since the dotted rectangle is often difficult to spot):

MarqueeStyle = 2 - Highlight Cell

This style inverts the current cell completely, making it very visible.   Values of EditBackColor and EditForeColor should be chosen carefully to make a pleasing effect if your grid is editable:

MarqueeStyle = 3 - Highlight Row

The entire row will be highlighted, but it won't be possible to tell which cell is the current cell in the row:

MarqueeStyle = 4 - Highlight Row & Cell

This value should only be used if 3D lines are used in the grid, since cell highlighting is accomplished using a "raised" appearance for the current cell:

MarqueeStyle = 5 - None

This setting will make the marquee disappear completely.   Often this setting is useful for cases where the current row is irrelevant, or where you don't want to draw the user's attention to the grid until necessary.

Using TrueGrid's Multiple Selection Feature

🔹 Multiple selection overview
🔹 How the user selects multiple rows
🔹 Working with the list of marked rows in your programs
🔹 Clearing marked rows using Visual Basic code

Multiple Selection Overview

TrueGrid supports a powerful and convenient means for selecting multiple rows of data and working with them in your programs.   Here are some important features of multiple selections:

1. Users can select and deselect rows directly using the mouse.

2. The BookmarkList property contains a complete list of all currently selected rows.

3. You can modify the BookmarkList directly, and the grid responds by showing or clearing the highlight on selected rows.

4. The SelectMode property gives you control over how the selection appears and operates.

The SelectMode property has 4 possible values:

0 - Disabled.   This is the default.   The user cannot select rows using the mouse.   However, if there is an existing selection present (such as one created by adding to BookmarkList), then it will be displayed.

1 - Enabled.   The user can select rows using the mouse, and any selected rows are displayed.

2 - Disabled & Hidden.   Like Disabled, the user cannot select rows.   In addition, any existing selection will not be visible.   However, the selection is maintained internally.

3 - Enabled & Hidden.   An odd setting, where the user may select rows, but they won't be shown.   If you find a use for this, let us know.

You can change the SelectMode at runtime in order to alternately show and hide the current selection.

How the User Selects Multiple Rows

For the user, the selection process is easy.   To select a row, the user moves the mouse into the leftmost margin of the grid until a small checkmark appears.   At that point, they can select a row, or deselect an already selected row.

Here, a grid appears with the mouse in the selection area and two rows (the first and third) selected):

[Picture of selection omitted to save space for demo help file.]

Notice how the color scheme chosen using SelectedForeColor and SelectedBackColor does not conflict with the existing color scheme or marquee presentation.   Making such choices carefully is important to assure that your user interface is easy to understand.

Unless you disable user selection (using SelectMode) the user can always select or deselect records, whether they were selected by the program (using BookmarkList) or whether they were originally selected by the user.   This can be very useful.   For example, an initial selection set can be highlighted using a query, then the user can add and remove records at their discretion.

Working with the List of marked Rows in your Programs

The grid maintains a list of all selected rows in the BookmarkList property.   This property contains a list of bookmarks, each corresponding to one of the selected grid rows.   The BookmarkCount property indicates how many bookmarks are currently in the list (and hence how many rows are selected).

You can use the BookmarkList to monitor the selection being made by the user, or to add additional selected rows of your own.   For example, once a selection is made, you could use the following code to go through the marked rows one at a time and perform some operation on each:

```
For i% = 0 To Table1.BookmarkCount - 1
```

```
    ' Position to the row
    Data1.RecordSet.Bookmark = Table1.BookmarkList(i%)

    ' Process the current row, then move on
    ProcessCurrentRecord Data1
Next i%
```

Note that changing the Bookmark property for the current data control's RecordSet will have the side effect of positioning the grid to the new location.   You may want to create a *clone* of the record set so you can process each record without affecting the rest of your application.

You can add bookmarks to the list of bookmarks for a grid, which will cause them to be highlighted.   To do this, you assign to the BookmarkList using an index one higher than the highest bookmark:

```
Sub HighlightManagers()
    ' Set criteria for FindFirst/FindNext
    criteria$ = "TITLE = " + chr$(34) + "Manager" + chr$(34)

    ' Loop through, adding each manager to the BookmarkList
    Data1.RecordSet.FindFirst criteria$
    While Not Data1.RecordSet.EOF
        Table1.BookmarkList(Table1.BookmarkCount) =
Data1.RecordSet.Bookmark
        Data1.RecordSet.FindNext criteria$
    Wend
End Sub
```

Clearing Marked Rows using Visual Basic Code

You can use the BookmarkList to clear selected rows as well:

1. To clear a single row, set the corresponding element in the BookmarkList property array to an empty string ("").   When you do this, the grid decrements the BookmarkCount property automatically and moves all subsequent bookmarks down by one index position, thus removing the deleted bookmark.   At the same time, any highlighted rows corresponding to the bookmark will no longer be highlighted.

2. You can clear the entire selection by setting BookmarkCount to zero (0).

Using TrueGrid in Callback (Application) mode

■ Introduction to TrueGrid Callback mode
■ Steps to create a Callback mode application
■ Creating a user friendly callback program

Introduction to TrueGrid Callback mode

When used in "Application (or Callback) mode"   (i.e., the DataMode property of the grid is set to 1) the grid is not connected to a database file.   Instead, data in the grid will be supplied and maintained by the programmer while the grid takes care of all user interaction and data display.

For example, the user may cover the grid (either completely or partially) with another window and later uncovers it.   The grid will be completely responsible to determine when and what portion of the grid need to be repainted, while the programmer does not need to know about any user interaction or display requirement at all.

The programmer needs only to concentrate on maintaining his data.   When the grid needs to display data on the screen, it will ask the programmer for it via the Fetch event.   Conversely, after user has made changes to the data in the grid, the grid will notify the programmer of the changes via the Update event so that the programmer can update his data to reflect the changes.   If data is changed internally rather by user editing on the grid, the programmer should notify the grid to repaint either by using the RefreshRow or RefreshColumn property, or by using the Refresh method.

The programmer should also make use of other TrueGrid events to make his application user friendly. These techniques are illustrated in the sections below and in the dBTable sample application that is included which with the TrueGrid product.

Steps to Create a Callback Mode Application

The basic steps for creating a TrueGrid application using callback mode are described below:

1. Determine method to store data.   The programmer is responsible to maintain and store data to be displayed on the grid.   The most common way is to store the data in Visual Basic arrays.   If the grid is very large (say, 1,000,000 rows by 5,000 columns), there may not be sufficient system memory to store the arrays and the programmer then needs to store the data in files or other storage media.   (When that happens, databases are usually used to store the data and the programmer reverts to using the Database mode of the grid.)   Another possibility is data displayed in the grid depends on some formula (usually a function of row and column numbers). The programmer then defines the data formula in the Fetch event.

2. Initialize the data and the grid.   As soon as the grid is loaded it will start asking the programmer (via the Fetch event) for data to display.   So the programmer should initialize the data before the grid is ready.   A good place to do this is either the **Main** procedure (if you start you program with **Main**) or the **Form_Load** procedure (for the Form which contains the grid).   The programmer then has to initialize some grid properties.   Some of the grid properties can be initialized at design time using either the properties window or the TrueGrid Layout Editor.   We will illustrate below how to initialize some properties at runtime.   Assuming you have a 3 rows by 2 column table entry and you store the data using 3 arrays (one array per row), the following code initializes the data and grid properties:

```
' Declare grid data as global array in some global module
Global Row1(1 to 2) As String
Global Row2(1 to 2) As String
Global Row3(1 to 2) As String

' Initialize array data in Main() or Form_Load()
For i% = 1 to 2
   Row1(i%) = "Row 1, Col" + Str$(i%)
   Row2(i%) = "Row 2, Col" + Str$(i%)
   Row3(i%) = "Row 3, Col" + Str$(i%)
```

```
      Next
      ...
      ' Assuming the control name of the grid is Table1, initialize
      ' grid properties in Form_Load():
      ' Define column heading text.  This can be done in code at
      ' runtime or in the Layout Editor during design time.
      Table1.ColumnName(1) = "Col 1"
      Table1.ColumnName(2) = "Col 2"

      ' Maximum number of characters allowed for columns.  (i.e.,
      ' User can enter no more than the number of characters
      ' specified below.)
      Table1.ColumnSize(1) = 16
      Table1.ColumnSize(2) = 16

      ' Columns display widths (in characters).  ColumnWidth are
      ' usually bigger than ColumnSize so that the table will look
      ' nice and will not be crowded.
      Table1.ColumnWidth(1) = 20
      Table1.ColumnWidth(2) = 20

      ' Column styles.  Specify left, center, or right justified.
      ' Also specifies if a column is read-only (i.e., user cannot
      ' edit that column).  The column style constants are defined
      ' in TGCONST.TXT (in your TrueGrid installation directory).
      Table1.ColumnStyle(1) = GRS_LEFT + GRS_READONLY
      Table1.ColumnStyle(2) = GRS_RIGHT + GRS_READONLY

      ' Define number of rows in the table.
      Table1.Rows = 3

      ' Initialize current cell position to upper left corner:
      Table1.RowIndex = 1
      Table1.ColumnIndex = 1
```

Note that the grid originally has zero number of columns, as you start setting the 1st, 2nd, column properties, etc., the number of columns automatically increases. Alternatively, you can set the number of grid columns by saying:

```
      Table1.Columns = 2
```

3. Define the Fetch event.   After the grid is loaded, it will start asking the programmer for data to display.   This is done via the Fetch event.   The programmer feeds data to the grid according to the Row and Col values:

```
      Sub Table1_Fetch (Row As Long, Col As Integer, Value As String)
         If Row =1 Then
               Value = Row1(Col)
         Else
               Value = Row2(Col)
         End If
      End Sub
```

The programmer cannot make any assumption as to when the grid will ask for data.   Also, the programmer cannot make the assumption that if data for a certain Row/Col is asked once, the grid will not ask for it again.   In short, it's the programmer's responsibility to store and maintain the data, and it's the grid's responsibility to display them properly.   This technique will free the programmer from worrying when and how to display data in the grid.

4. Define the <u>Update</u> event.   After the user has finished editing data in a cell and the data has been validated by the programmer via the <u>Validate</u> event, the grid will then notify the programmer the cell now has a new value via the Update event.   In this event, the programmer uses the value supplied by the grid to update the data he or she is responsible for storing and maintaining:

```
Sub Table1_Update (Row As Long, Col As Integer, Value As String)
   If Row =1 Then
         Row1(Col) = Value
   Else
         Row2(Col) = Value
   End If
End Sub
```

5. Refresh the grid if data has been changed by the program.   If data is changed by the program instead of by the user editing on the grid, the programmer must notify the grid of the changes so that the grid will properly display the updated data.   This can be done by one of the following three methods:

```
' Using the RefreshRow property to repaint a row (in this
' example, row number 2):
Table1.RefreshRow = 2

' Using the RefreshColumn property to repaint a column (in
' this example, column number 1):
Table1.RefreshColumn = 1

' Using the Refresh method to repaint the entire grid:
Table1.Refresh
```

Creating a User Friendly Callback Program

This section describes using three TrueGrid events to assist user in the data entry process and thus creating a user friendly table entry program.   The events used in this application are only some of the TrueGrid events available.   After learning this application, you should familiarize yourself with all available TrueGrid events and start using them to create your own user-friendly applications.   Also, examine the dBTable sample application for more details.

1. **Using the KeyPress event.**   The programmer can use **KeyPress** event to restrict and modify user key input.   The following example converts the user key entry to upper case letters, and restrict the user entry to letters and digits only.

```
Sub Table1_KeyPress (KeyAscii As Integer)
   ' Convert key to upper case
   ToUpper KeyAscii

   ' Cancel user key input if it is not a letter or a digit
   If (KeyAscii < 65 And KeyAscii > 90) And (KeyAscii < 48 And
KeyAscii > 57) Then
         KeyAscii = 0
   End If
End Sub
```

2. **Using the <u>Validate</u> event.**   Before moving to another grid cell, this routine lets the programmer examine the value in the current cell.   If the value entered is invalid, the programmer can (optional) display a message, then set Cancel to True to prohibit the user moving on to another cell.   The Validate and KeyPress events together provide a very easy way for programmers to check, restrict and validate user input in TrueGrid.

```
Sub Table1_Validate (Row As Long, Col As Integer, Value As String,
Cancel As Integer)
   Select Case Row
```

```
              Case 1
                    ' Valid if starts with "Row 1"
                    If Left(Value, 5) = "Row 1" Then Exit Sub
              Case 2
                    ' Valid if start with "Row 2"
                    If Left(Value, 5) = "Row 2" Then Exit Sub
        End Select

        ' Display warning message for user
        MsgBox "Invalid Value"

        ' Data validation fails, prohibit user from moving to
        ' another cell
        Cancel = True

    End Sub
```

3. **Using the <u>ColumnChange</u> event.**   User can use the _ColumnChange() event to display a different help string when the user is at different table column.   This is to make the program user friendly by assisting the user during the data entry process.

```
    Sub Table1_ColumnChange ()
        ' When this event is called, ColumnIndex has the new column
        ' position value
        Select Case
              Case 1
                    ' Display help string at Label1
                    Label1.Caption = "Please enter value for Column 1"
              Case 2
                    ' Display help string at Label1
                    Label1.Caption = "Please enter value for Column 2"
              Case 3
                    ' Display help string at Label1
                    Label1.Caption = "Please enter value for Column 3"
        End Select
    End Sub
```

TrueGrid Properties

TrueGrid Properties are used to define a state or characteristic of the grid control.   Properties can be set at design time via the properties window.   The contents of the properties can be referenced, and usually modified at runtime also.   Here is the list of the grid control properties:

Active	ColumnVisible	HorzColor	Rows
BookmarkCount	ColumnWidth	HorzLines	SelectedBackColor
BookmarkList	Configurable	Hwnd	SelectedForeColor
BottomRow	DataChanged	HwndEdit	SelectMode
ColumnChanged	DataMode	InactiveBackColor	SelLength
ColumnField	DataSource	InactiveForeColor	SelStart
ColumnIndex	Editable	Layout	SelText
ColumnName	EditBackColor	LeftColumn	Text
ColumnOrder	EditForeColor	MarqueeStyle	TopRow
Columns	EditMask	RefreshColumn	VertColor
ColumnSize	HeadBackColor	RefreshRow	VertLines
ColumnStyle	HeadForeColor	RightColumn	
ColumnText	Headings	RowIndex	

Standard Properties

◼ Grid Events

These event functions can be associated with a grid control and are used to respond to database events. Events enable the programmer to specialize the actions of a grid control.

Append

CellChange

Change

ColumnChange

Fetch

RowChange

Update

Validate

Standard Events

Grid Active Property

Usage   *grid.***Active** = *boolean*

Read/Write at design time and run time.
Valid only in database mode; not used in callback mode.

Description   This property determines if the grid will respond to changes made to the database using other database functions (either in code, or via other bound controls).   When *True*, the grid will refresh itself automatically when the record pointer is moved (using commands like Data1.Recordset.MoveNext) or when records are added, deleted, updated, sorted, or when selections are made using SQL statments.   When set to *False*, the grid will not refresh itself automatically.

Normally, you should leave this property set to *True*, and you should rarely need to change it.   Even when Active is set to *True*, the grid automatically suppresses updates to the grid *while your program code is executing*.   Thus, you can perform any number of complex operations, and the grid will remain unchanged until your program becomes idle or until you send the Refresh method to the grid.   Because of this automatic behavior, you should rarely need to use the Active property.

Once case where it is useful to set Active to false is when you are displaying a modal dialog and a grid is present elsewhere on the screen.   For example, if you were to create a modal dialog which will calculate totals for columns, the grid would normally track the record you are currently positioned to, which may be undesirable.   In such circumstances, you might want to set Active to *False* so it remains dormant until the dialog completes its task.

This behavior is different from bound text boxes, which update instantly even while your program is running.

When set to *False*, the grid can still be used interactively.   However, the data displayed may not be the actual data in the visible records.   This situation can be very confusing to the user and should be avoided.   As soon as Active is set to *True*, the grid will refresh itself if necessary, then remain in sync with the database thereafter.

Grid BookmarkCount Property

Usage *grid.***BookmarkCount** = *numericexpression*

Read/Write at run time.   Not available at design time.

Description This property contains the number of rows currently selected.   The current highlighted row is not considered a selected row.   Only rows selected by the user using the checkmark cursor, or added to the BookmarkList directly are considiered "selected". Row selection is controlled by the setting of the SelectMode property.

As the user selects rows, BookmarkCount increases to reflect the current selection size. Adding selected bookmarks to the BookmarkList will also cause the BookmarkCount to increase.

You can only set the value of BookmarkCount to zero (0), which clears the BookmarkList and any selected rows on the grid.   Any other value is illegal.

See Also For a complete discussion of multiple selection, with examples, see Multiple Selection Overview.

Grid BookmarkList Property

Usage *grid.***BookmarkList(***index***)** = *bookmarkstring*

Read/Write at run time.   Not available at design time.

Description The BookmarkList property contains a list of bookmarks for all selected grid rows.   When the user selects rows, their bookmarks are automatically added to the BookmarkList and <u>BookmarkCount</u> is automatically incremented.   The ability to select rows is controlled by the <u>SelectMode</u> property.

The *index* value is zero-based and has a maximum value one less than the current value of BookmarkCount.

All bookmarks in the list can be read, but you can only add bookmarks to the end of the list:

```
tb.BookmarkList(tb.BookmarkCount) = Data1.RecordSet.Bookmark
```

You can remove an individual bookmark (and unhighlight it in the grid) if you know the bookmark index.   This is done by setting the particular value in the list to an empty string:

```
tb.BookmarkList(5) = ""
```

The above line of code removes the bookmark for the 5th selected row.   The selection highlighting will disappear, the BookmarkCount will be decreased by one, and any bookmarks higher in the list will be moved down by one (filling the slot vacated by bookmark item 5).

See Also For a complete discussion of multiple selection, with examples, see <u>Multiple Selection Overview</u>.

Grid BottomRow Property

Usage   *numericresult = grid.***BottomRow**

Read-only at run time.   Not available at design time.

Description   This property denotes the one-based index of the bottommost grid row.   The difference between this property and the TopRow property, plus 1, is the number of records currently visible in the grid.   If there are fewer rows of data than screen rows, BottomRow will refer to the bottom row of data, not the bottom screen row.

To the grid, the term row number refers to the row within the overall set of data being browsed, rather than the screen row number.   Whether in callback or database mode, row 1 does not refer to the topmost row of the displayed grid, but refers to the first row of data.   A grid may contain thousands or even millions of rows.

Grid ColumnChanged Property

Usage	*boolean* = *grid*.**ColumnChanged(***column***)**
	Read-only at run time. Not available at design time.
Description	The ColumnChanged property returns *True* if the data in the current row of the specified column has been edited by the user, but has not yet been updated to the database. This happens if a cell has been edited, but the current row position hasn't yet been moved, or if it is about to be moved and a Validate handler is executing.
	Cells which have been changed, but not yet updated, appear in red in the grid until they are written to the database.
	The <u>DataChanged</u> property is always *True* when one or more columns have a *True* value for their ColumnChanged property. Setting DataChanged to *False* has the side effect of changing all ColumnChanged values to *False* as well.
	This property is especially useful in data control Validate handlers, and used in conjunction with the <u>ColumnText</u> property.
See Also	<u>Data Validation Techniques</u>

Grid Columns Property

Usage   *grid.***Columns** = *numericexpression*

Read/Write at run-time.   Not available at design time.

Description   This property returns the number of columns currently defined for the grid.   This value reflects the number of columns which can potentially be displayed, but does not reflect the number of actual columns displayed.   This depends upon whether columns are visible (using the <u>ColumnVisible</u> property) and whether they are scrolled off the display on the right or left.

In callback and database modes, columns are created automatically when properties such as <u>ColumnName</u> are used to assign characteristics to a new column.

The value of Columns may be set to zero, which will physically remove all column information from the grid's memory.   All data and headings will be cleared, and the <u>ColumnIndex</u> property will subsequently return zero, indicating that there is no data on the grid.

Attempting to set Columns to a non-zero value is not allowed and will cause a Visual Basic error.

Grid ColumnField Property

Usage   *grid.***ColumnField(***column***)** *= stringexpression*

Read/Write at run time.   Access via Layout Editor at design time.

Description   This property holds the database field name for a grid column.   It will be set automatically if <u>Layout</u> is chosen while a valid <u>DataSource</u> is present at design-time.   It is usually not needed in callback mode.

This property is useful for adding new columns to an existing grid layout when the structure of its associated database has been modified.

When assigning a value to this property, you can use a column index one greater than the maximum number of columns currently contained in the grid (returned by the <u>Columns</u> property).   The grid will create a new column and automatically increase the column count.

Side Effects   If the ColumnField property is set, the column will be refreshed with data from the new field.

Grid ColumnIndex Property

Usage	*grid.***ColumnIndex** = *numericexpression*
	Read/Write at run time. Not available at design time.
Description	This property denotes the one-based index of the current cell's column position. It may be set at run time to highlight a different cell within the current row. If the column is visible, the <u>marquee</u> will be moved to the selected column. If the column is not visible, it will be made visible as as a result of setting this property.
	Immediately after a new grid is created, there is no layout, so this property has a value of 0, even though technically this value is out of range.
Side Effects	The new column will become visible (which may have an effect on the <u>LeftColumn</u> property). In any case, the marquee will be move to the selected column.

Marquee

A dotted box, highlighted cell, or highlighted row around the perimeter of the current grid cell.

See also

MarqueeStyle

Grid ColumnName Property

Usage   *grid.***ColumnName(***column***)** = *stringexpression*

Read/Write at run time.　Access via Layout Editor at design time.

Description   This property holds the heading text for a grid column.　It is set to the appropriate attribute name automatically when the <u>Layout</u> property is selected and <u>DataSource</u> is set to a properly specified database.　It may also be set at run time in either callback or database mode.

In callback mode, a value should be supplied if headers are desired, either via the Layout Editor or in code at run time.

The Layout Editor may be used at design time to change the default values used in automatic database mode, or to assign new values for callback mode.

When assigning a value to this property, you can use a column index one greater than the maximum number of columns currently contained in the grid (returned by the <u>Columns</u> property).　The grid will create a new column and automatically increase the column count.

Side Effects   This call will automatically refresh the headings to reflect the changed column name.

Grid ColumnOrder Property

Usage	*grid*.**ColumnOrder** = *numericexpression*
	Read/Write at run time. Column order can be specified in the Layout Editor at design time.
Description	This property holds the one-based sequence number for a grid column (i.e., the position of the column within the grid layout). In database mode, columns are ordered according to the database field layout.
	At run time, this property array is changed automatically when the user shuffles columns with the mouse. You can change this property using code, which causes the columns to appear in a different order in the grid.
	When you refer to a column element in code, the order of the columns in the grid does not need to be known. For example, referring to ColumnName(1) will always access the same column name, regardless of whether the columns have been moved by the user or whether their ColumnOrder property has been changed. This allows code to be consistent while still permitting users to customize column positions to their liking.
	ColumnOrder <u>example</u>
Side Effects	The columns will be repositioned on the display if necessary.

ColumnOrder Example

The grid contains five columns: **NAME, ADDRESS, CITY, STATE, ZIP**.   The user drags the **STATE** column to the first column position.   The ColumnOrder array now looks like this:

```
ColumnOrder(1) = 2  ' NAME
ColumnOrder(2) = 3  ' ADDRESS
ColumnOrder(3) = 4  ' CITY
ColumnOrder(4) = 1  ' STATE
ColumnOrder(5) = 5  ' ZIP
```

To achieve the same result in code, you would have to execute the first four of these assignment statements.   By itself, the statement

```
Table1.ColumnOrder(4) = 1
```

 merely exchanges the **NAME** and **STATE** columns.

Grid ColumnSize Property

Usage   *grid.***ColumnSize(***column***)** *= numericexpression*

Read/Write at run time.   Not available at design time.

Description   This property holds the database width, in bytes, for each grid column.   It is set to the appropriate field width (not display width) automatically when the <u>DataSource</u> property is set and the <u>Layout</u> property is chosen for the first time.   It may also be set at run time in either callback or database mode.

This property does not affect the grid layout, but imposes a limit on the number of characters that may be entered when editing a cell.   If set to zero, no such limits are imposed.

If the value of this property is 256 or greater, then the column in question cannot be edited.   Even if this value is zero (as is the case with native-mode fields), editing will be disabled on a per cell basis if the field contains 256 or more characters.

In database mode, if the database does not supply a width value, then no limits are imposed.   In either callback or database mode, a value may be supplied in code at run time in order to restrict the amount of data the user can enter.

When assigning a value to this property, you can use a column index one greater than the maximum number of columns currently contained in the grid (returned by the Columns property).   The grid will create a new column and automatically increase the column count.

Grid ColumnStyle Property

Usage   *grid.***ColumnStyle(***column***)** *= integerexpression*

Read/Write at run time.   Access via Layout Editor at design time.

Description   This property determines justification and editability for a grid column.   By default, grid columns are left-justified and editable, with the exception of numeric columns, which are right-justified and editable.   This property is set to the appropriate value when the <u>DataSource</u> property is set and the <u>Layout</u> property is selected for the first time.   It may also be set at run time in either callback or database mode.   The Layout Editor may be used at design time to change the default database mode values, or to assign new values for callback mode.

The following constants can be used to set the value of this property:

```
Const GRS_LEFT = 0        ' Left justify column
Const GRS_CENTER = 1      ' Center column
Const GRS_RIGHT = 2       ' Right justify column
Const GRS_READONLY = 8192 ' Disable editing for column
```

When setting the ColumnStyle property, use one of the first three values, optionally Or'ed with the last one.   When assigning a value to this property, you can use a column index one greater than the maximum number of columns currently contained in the grid (returned by the <u>Columns</u> property).   The grid will create a new column and automatically increase the column count.

See Also   <u>ColumnSize</u>, for a discussion of editability of potentially large items.

Side Effects   The column will automatically be refreshed when this property is changed.

Grid ColumnText Property

Usage   *string* = **ColumnText(***column***)**

Read-only at run time.   Not available at design time.

Description   This property returns the cell text of the current row of the specified column.   The column number must be between 1 and the maximum number of columns (specified by <u>Columns</u>).

If the cell text has been edited, then the edited contents are returned rather than the actual value of the data in the database.

This property is typically used in conjunction with <u>ColumnChanged</u> in order to perform data validation on the entire modified grid row.

See Also   <u>Data Validation Techniques</u>

Grid ColumnVisible Property

Usage	*grid.***ColumnVisible(***column***)** = *boolean*
	Read/Write at run time. Not available at design time.
Description	This property determines whether a given column is actually displayed. At run time, when the user resizes a column down to nothing, this property is set to *False* for the corresponding column, but the prior width is retained. There is no way for the user to get the column back unless the application developer provides a menu item or button that resets this property to *True*.
	In layout mode, columns which are hidden are physically removed from the layout and cannot be retrieved unless the layout editing session is canceled and the changes discarded.
	When assigning a value to this property, you can use a column index one greater than the maximum number of columns currently contained in the grid (returned by the <u>Columns</u> property). The grid will create a new column and automatically increase the column count.
Side Effects	The grid will be automatically refreshed to show or hide the chosen column.

Grid ColumnWidth Property

Usage   *grid.***ColumnWidth(***column***) =** *numericexpression*

Read/Write at run time.   Access via Layout Editor at design time.

Description   This property holds the display width, in character units, for each grid column.   It is set to the appropriate value automatically when the <u>DataSource</u> property is set and the <u>Layout</u> property is selected for the first time.   It may also be set at run time in either callback or database mode.

The actual column width depends upon the current font.   For fixed-width fonts, columns will hold exactly as much data as ColumnWidth specifies.   For proportional fonts, the unit of measurement is the average character width for the current font.

The maximum number of columns currently contained in the grid (returned by the <u>Columns</u> property).   The grid will create a new column and automatically increase the column count.

Side Effects   When this property is changed, the grid will automatically reflect the changes.

Grid Configurable Property

Usage   *grid.***Configurable** *= boolean*

Read/Write at design time and run time.

Description   This property determines whether the user can move and resize grid columns at run time. By default, both of these operations are permitted.   If this property is changed while the user is resizing or adjusting columns, then it will not take effect until the user has completed the current operation.

Grid DataChanged Property

Usage   *grid.***DataChanged** = *boolean*

Read/Write at run time.   Not available at design time.

Description   When a data control moves from record to record, the grid automatically moves its highlighting to the new row and DataChanged is set to *False*.   If the data in the grid has been changed in any way other than moving to a different record, DataChanged becomes *True*.

Any grid data which has been changed is highlighted in red to indicate that it hasn't been written to the database.   If DataChanged is *True*, you can inspect the values of ColumnChanged and ColumnText to determine the exact nature of the changes.

When the data control starts to move to a different record, the Validate event occurs. Then if DataChanged is True for the grid, the data control invokes the Edit and UpdateRecord methods.   Finally, data from the grid is saved to the database.

In the code for the Validate event, you can set this property to *False* where you do not want to save the data in the database.   When set to *False*, any red highlighting which indicated that the user has changed data disappears and the original data will appear in its place.

See Also   Data Validation Techniques

Grid DataMode Property

Usage	*grid.***DataMode**
	Read-only at run time.
Values	0 - Database Mode (default)
	1 - Callback, or application Mode
Description	This property determines whether the grid fetches cell data from a database or from the application (via the Fetch event).

The phrase "database mode" means that DataSource is 0.   The phrase "callback mode" means that DataSource is 1.

When this property is set to 1, the grid operates in callback mode, in which it sends the <u>Fetch</u> event to obtain cell data.   This allows grids to be used to browse and edit Visual Basic arrays, DOS files, or other application-defined data.

When the property value is zero, the grid manages all database operations completely. In such cases, the database in use is determined by the setting of <u>DataSource Property</u>. In database mode, the Fetch event is never sent.

This property cannot be changed at runtime, so grids must operate in one mode or the other at all times.

Grid DataSource Property

Usage   Available only at design time.

Description This property is used to bind the TrueGrid control to a data control.   It must be set at design time, and cannot be changed while the program is running.   In order to set DataSource, a data control must already be present on the same form as the grid control.

Once the DataSource property is set, clicking on the <u>Layout</u> property will ask whether the database layout should automatically be used by the grid.

Grid Editable Property

Usage	*grid.***Editable** = *boolean*
	Read/Write at design time and run time.
Description	This property determines whether the contents of the grid can be edited directly. By default, they can.
	If this property is *True*, it may be overridden by the ColumnStyle property for an individual column if the GRS_READONLY flag is in effect. However, if this property is *False*, then no editing events will be accepted for any column, regardless of ColumnStyle.
	If the user is editing the current cell when this property is changed, the edit will continue as usual. However, no further editing will be permitted after the current edit is complete.
	If the data control associated with the grid has its ReadOnly property set to *True*, then editing will not be permitted, even if this property is set to *True*. However, the value of the Editable property itself will not be affected.
See Also	ColumnSize, for a discussion of the editability of potentially large items.
Ideas	Use a one-column grid in callback mode with the Editable property set to TRUE to provide a list box in which the user can edit items directly.

Grid EditBackColor Property

Usage   *grid.***EditBackColor** = *colorvalue*

Read/Write at design time and run time.
Default is System Highlight Color (0x8000000D)

Description   This property denotes the background color of the grid's editing window when editing is in
progress.   If editing is in progress, the color of the edit area will not be changed until the
current edit has been completed.

You can set this color by clicking on the ellipsis in the properties window .   A color palette
will appear which allows you to select a color.   You can also type the hexadecimal value
of a color directly into the settings box.

Grid EditForeColor Property

Usage   *grid.***EditForeColor** = *colorvalue*

Read/Write at design time and run time.
Default is System Highlight Text Color (0x8000000E)

Description   This property denotes the color of the grid's editing window text when editing is in progress.   If editing is in progress, the color of the edit area will not be changed until the current edit has been completed.

You can set this color by clicking on the ellipsis in the properties window.   A color palette will appear which allows you to select a color.   You can also type the hexadecimal value of a color directly into the settings box.

Grid EditMask Property

Usage   *grid.***EditMask(***column***) =** *stringexpression*

Read/Write at run time.   Access via Layout Editor at design time.
Ignored in callback mode.

Description   The EditMask property allows you to specify a <u>Visual Basic Format String</u> which will be used for formatting all data displayed in the specified column.   At design type, you can use the <u>Layout</u> property to set a pre-defined value for this edit mask.

The format string will be used only for *displaying* data.   Data input will occur in unformatted mode and the data will be reformatted when editing is complete.   If you do not include an edit mask, or you set the EditMask property to an empty string, then data will be displayed in the standard Visual Basic string representation for the field.

Grid HeadBackColor Property

Usage   *grid.***HeadBackColor** = *colorvalue*

Read/Write at design time and run time.
Default is System Active Caption Color (0x80000002)

Description   This property denotes the background color of the grid's headings.

This color will be displayed only while the grid has focus.   When the grid loses focus, the heading colors are determined by the current setting of <u>InactiveForeColor</u> and <u>InactiveBackColor</u>.

You can set this color by clicking on the ellipsis in the properties window.   A color palette will appear which allows you to select a color.   You can also type the hexadecimal value of a color directly into the settings box.

Side Effects   The grid headings will automatically be redrawn in the new color.

Grid HeadForeColor Property

Usage	*grid.***HeadForeColor** = *colorvalue*
	Read/Write at design time and run time.
	Default is System Caption Text Color (0x80000009)
Description	This property denotes the color of the grid's heading text.
	This color will be displayed only while the grid has focus. When the grid loses focus, the heading colors are determined by the current setting of InactiveForeColor and InactiveBackColor.
	You can set this color by clicking on the ellipsis in the properties window. A color palette will appear which allows you to select a color. You can also type the hexadecimal value of a color directly into the settings box.
Side Effects	The grid headings will automatically be redrawn in the new color.

Grid Headings Property

Usage   *grid.***Headings** = *boolean*

Read/Write at design time and run time.

Description   This property determines whether the grid displays column headings.   Setting this property to *False* allows an additional row of data to be displayed.

Side Effects   The grid will be redrawn in its entirety to effect the change.   Setting this property to *False* disables column movement, since you can only move a column by grabbing its header. Column resizing is not affected. Setting this property to *False* will also erase the vertical scroll bar if all records become visible.

Grid HorzColor Property

Usage	*grid.***HorzColor** = *colorvalue*
	Read/Write at design time and run time. Default is System Window Frame Color (0x80000006)
Description	This property denotes the color of the grid's horizontal lines. This color is only meaningful if horizontal lines are turned on, via the HorzLines property.
	You can set this color by clicking on the ellipsis in the properties window. A color palette will appear which allows you to select a color. You can also type the hexadecimal value of a color directly into the settings box.
Side Effects	The grid headings will automatically be redrawn in the new color.

Grid HorzLines Property

Usage *grid.***HorzLines** = *numericexpression*

Read/Write at design time and run time.

Values 0 - None (default)
1 - Single
2 - 3D

Description This property determines whether the grid displays horizontal lines between rows.

If the value of this property is 0, then no lines are displayed.   If the value is 1, then single-width lines are displayed in the color specified by HorzColor.

If the value of this property is 2, then the grid will attempt to draw the lines in 3D mode. The grid uses the HorzColor value for the darker border of the 3D "groove" between cells, and chooses another color from the available colors in the system palette.   Some experimentation is usually required to achieve the desired effect.   The standard grey 3D effect is chosen by selecting a dark grey color for the lines and a medium grey for the grid background color.

Side Effects The grid will be redrawn in its entirety to effect the change.   Changing this property to a non-zero value may decrease the number of lines which may be displayed (since horizontal lines take up a bit more space).

Grid InactiveBackColor Property

Usage   *grid.***InactiveBackColor** = *colorvalue*

Read/Write at design time and run time.
Default is System Inactive Caption Color (0x80000003)

Description   This property denotes the background color of the grid's headings whenever the grid does not have focus.

When the grid does have focus, the heading color is controlled by the <u>HeadForeColor</u> and <u>HeadBackColor</u> properties.

You can set this color by clicking on the ellipsis in the properties window.   A color palette will appear which allows you to select a color.   You can also type the hexadecimal value of a color directly into the settings box.

Side Effects   The grid headings will automatically be redrawn in the new color if the grid currently does not have focus.

Grid InactiveForeColor Property

Usage   *grid.***InactiveForeColor** = *colorvalue*

Read/Write at design time and run time.
Default is System Inactive Caption Text Color (0x80000013)

Description   This property denotes the color of the grid's heading text whenever the grid does not have focus.

When the grid does have focus, the heading color is controlled by the HeadForeColor and HeadBackColor properties.

You can set this color by clicking on the ellipsis in the properties window.   A color palette will appear which allows you to select a color.   You can also type the hexadecimal value of a color directly into the settings box.

Side Effects   The grid headings will automatically be redrawn in the new color if the grid currently does not have focus.
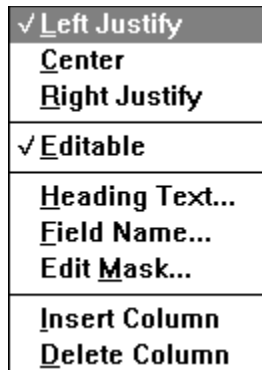
Grid Layout Property

Usage   This property is available only at design time.   Clicking on the Layout propety in the property bar, or clicking over a grid with the right mouse button will bring up the Layout Editor, which is used to configure per-column information in the grid.

Description   The Layout property is a vehicle for manipulating the column layout of the grid.   The user is presented with a modal dialog box which allows the following property arrays to be changed at design time: ColumnField, ColumnName, ColumnOrder, ColumnStyle, ColumnWidth, EditMask.   The dialog box contains a replica of the grid in question, which may be manipulated with the mouse just like at run time.

Within the properties window, one of the following non-editable strings is displayed:

(none)	No layout information exists.
(database)	Default layout information was read from the DataSource property.
(modified-database)	
	The user made changes to a database layout using the Layout Editor.
(user-defined)	Layout created from scratch.

Within the Layout Editor, double-clicking a column (or single-clicking with the right button) pops up the following menu:

```
√ Left Justify
  Center
  Right Justify
─────────────
√ Editable
─────────────
  Heading Text...
  Field Name...
  Edit Mask...
─────────────
  Insert Column
  Delete Column
```

Each section of the menu has the following behavior:

-   One of the text justification options will always be checked.   These correspond to the GRS_LEFT, GRS_CENTER, and GRS_RIGHT values of the ColumnStyle property.

-   Editable will not be checked if the ColumnStyle GRS_READONLY flag is set.

-   The Heading Text and Field Name options are used to set the ColumnName and ColumnField properties.

-   The "Edit Mask" option allows you to specify a Visual Basic edit mask which will be used to format the contents of the column.   The edit mask uses a Visual Basic Format$ Mask and is used only for display purposes (input occurs with no mask).   For a complete description of this feature, see the EditMask property description.

-   Insert Column will create a new column immediately to the left of the column where the menu was first popped-up.   Delete Column will remove the corresponding column.   All changes are permanent (although the entire layout editing session may be discarded using the Discard option in the system menu for the Layout Editor).

If either the Heading Text or Field Name item is chosen, a small text editor appears within the column heading area.   Type in the new value and hit the Enter key to commit the change, the Esc key to discard it.

The Layout Editor does not have its own menu bar, but customizes the system menu as follows:

Move	
Close	Alt+F4
Discard	
Clear Fields	

The Close option will save all changes made during the editing session and return to the Visual Basic form.   The Discard option will forget all changes made during the editing session and return to the form. In the latter case, the original layout will remain unchanged.

The "Clear Fields" option will remove all fields from the current working layout (without disturbing the original).   If you close the layout editor after Clear Fields, then the grid will be initialized just as if it had never had a layout specified at all.

Side Effects   All column properties may be affected except ColumnSize, which cannot be set in design mode.   ColumnWidth is set implicitly when you resize a column with the mouse.

Grid LeftColumn Property

Usage   *grid.***LeftColumn** = *numericexpression*

Read/Write at run time.   Not available at design time.

Description   This property denotes the one-based index of the leftmost grid column.   You can use this property to scroll the grid horizontally in code at run time.   If the user scrolls the grid using the horizontal scroll bar, this property will contain the column number of the column which is currently at the left hand side of the grid.

Side Effects   If changed, the grid will be redrawn to display the new column positions.

Grid MarqueeStyle Property

Usage   *grid.***MarqueeStyle** = *numericexpression*

Read/Write at design time and run time.

Values   0 - Dotted Cell Border (Default)
1 - Solid Cell Border
2 - Highlight Cell
3 - Highlight Row
4 - Highlight Row & Cell
5 - None

Description   This property allows you to specify the method which the grid will use to highlight the current row and cell in the grid.   There are six possible values for this property:

0 - Dottec Cell Border
  The current cell within the current row will be highlighted by drawing a dotted border around the cell.   In Microsoft Windows terminology, this is usually called a *focus rectangle*.   This is the default value for this property.

1 - Solid Cell Border
  The current cell within the current row will be highlighted by drawing a solid box around the current cell.   This is more visible than the dotted cell border, especially when 3D line properties are used for the grid.

2 - Highlight Cell
  The entire current cell will be highlighted by inverting the colors within the entire cell. This provides a very distinctive block-style highlight for the current cell.

3 - Highlight Entire Row
  The entire row containing the current cell will be highlighted by inverting the colors within the row.   In this mode, it is not possible to visually determine which cell is the current cell, only the current row.   When the grid is not editable, this setting is often preferred, since cell position is then irrelevant.

4 - Highlight Row and Cell
  The entire row will be highlighted.   The current cell within the row will be "raised" so that it appears distinctive.   This setting doesn't appear clearly with all background and color settings.   The best effect is achieved by using 3D lines and a grey background.

5 - None
  The marquee will not be shown.

Changing this property at runtime has the effect of changing the cell highlighting for the specified cell or row.

Grid RefreshColumn Property

Usage   *grid.***RefreshColumn** = *numericexpression*

Write only at run time.   Not available at design time.

Description   This property allows a column of data to be refreshed without repainting the entire window.   If set to zero, the current column (<u>ColumnIndex</u>) is refreshed.   Otherwise, the specified column is refreshed.   If text editing is in progress, this call will have no effect on the text being edited, but will refresh the data in the other rows.   You can use the <u>Text</u> property to change the data in the cell being edited.

Typically, this call is useful only in callback mode, where the grid is unaware of changes made to the cell data.   In database mode, the column will be refreshed as well, but usually it is not necessary to do so.   However, if the grid's <u>Active</u> property is set to *False*, the refreshed column will be filled with data from the current column of the database, although the other columns may not reflect the change.

Grid RefreshRow Property

Usage   *grid.***RefreshRow** = *numericexpression*

Write only at run time.   Not available at design time.

Description   This property allows a row of data to be refreshed without repainting the entire window. If set to zero, the current row (RowIndex) is refreshed if it is visible.   Otherwise, the specified row is refreshed.   If text editing is in progress, this call will have no effect on the text being edited, but will refresh the data in the other columns.   You can use the Text property to change the data in the cell being edited.

Typically, this call is useful only in callback mode, where the grid is unaware of changes made to the cell data.   In database mode, the row will be refreshed as well, but usually it is not necessary to do so.   However, if the grid's Active property is set to FALSE, the refreshed row will be filled with data from the current row of the database, although the other rows may not reflect the change.

To TrueGrid, the term row number refers to the row within the overall set of data being browsed, rather than the screen row number.   Whether in callback or database mode, row 1 does not refer to the topmost row of the displayed grid, but refers to the first row of data.   A grid may contain thousands or even millions of rows.

Grid RightColumn Property

Usage   *grid.***RightColumn**

Read-only at run time.   Not available at design time.

Description   This property can be used to retrieve the one-based index of the rightmost grid column. The difference between this property and the <u>LeftColumn</u> property, plus 1, is the number of columns currently visible.

Grid RowIndex Property

Usage   *grid.***RowIndex** = *numericexpression*

Read/Write at run time.   Not available at design time.

Description   This property denotes the one-based index of the current row position.   It may be set at run time to highlight a different cell within the current column.   When set in this fashion, the specified row will be made visible if it is not already visible and the marquee will be moved to the appropriate cell.

If the current cell is being edited, then the contents of the cell will be verified and the edit will be completed before moving the cell pointer.   If the validation for the current cell fails (due to the Validate event), then the cell pointer will *not*   be changed and the RowIndex will remain the same as it was before this call was made.

If the value of RowIndex is zero, it indicates an empty selection set in database mode, or the absence of data in callback mode.

In database mode, the current value of the Rows property may not reflect the actual number of rows in the database.   In such cases, it is possible to set RowIndex to a value which is larger than the actual number of rows in the current database or subset.   If you set RowIndex to a value larger than the number of actual rows, the grid will attempt to position the database to the newly selected row.   If the set does not contain the given number of rows, the grid automatically sets Rows and RowIndex to their correct values.

To TrueGrid grid, the term row number refers to the row within the overall set of data being browsed, rather than the screen row number.   Whether in callback or database mode, row 1 does not refer to the topmost row of the displayed grid, but refers to the first row of data.   A grid may contain thousands or even millions of rows.

Side Effects The grid will be scrolled or repositioned as necessary to cause the current cell to become visible.   There are no side effects for accessing this value, only for setting it.

Ideas   Setting RowIndex to the same value it presently has, such as:

Grid1.RowIndex = Grid1.RowIndex

will cause the current cell to be validated and editing completed.   this is a useful trick to use if you want to force editing to completion.

Grid Rows Property

Usage	*grid.***Rows** = *numericexpression*
	Read/Write at run time. Not available at design time.
Description	This property defines the number of rows displayed by the grid. If set to zero, no data is shown, although column headings will be displayed if they are enabled and there is at least one column.
	To TrueGrid, the value of Rows refers to the number of rows in the overall set of data being browsed, rather than the number of rows displayed on the screen. Whether in callback or database mode, row 1 does not refer to the topmost row of the displayed grid, but refers to the first row of data. A grid may contain thousands or even millions of rows.
	In callback mode, the grid will be repainted if this property changes the number of rows which are visible on the display. In addition, the scrollbars will be adjusted to reflect the relative position of the current row with respect to the number of total rows. If the scrollbars are no longer needed they will be removed, and if not present, they will be made visible if now required.
	In database mode, this property is maintained by the grid and cannot be changed. Under many circumstances, however, this value will *not reflect the actual number of rows in the database.* This is because the grid itself does not attempt to determine how many database rows there are until absolutely necessary. Why? Because many queries (and database file formats) often require a great deal of work to determine how many rows there are.
	For example, if a database contains 10,000 records, and you perform a text search query, all the records in the database must be scanned to find the matching rows. If there are 1,000 matches, the grid would need to complete the entire query in order to set the Rows property to the correct value.
	Instead, the grid will simply set the row number to a very large value (currently 1,000,000) until it encounters the last record in the set, at which point it will change the Rows property to contain the correct value automatically.
Side Effects	The grid and scrollbars will be adjusted to reflect the new row count when this value is changed in callback mode.

Grid SelectedBackColor Property

Usage	*grid.***SelectedBackColor** = *colorvalue*
	Read/Write at design time and run time. Default is black.
Description	This property denotes the background color to be used for rows which have been selected by the user, or have been added to the <u>BookmarkList</u>. Selected rows will only appear if <u>SelectMode</u> is set properly.
	You can set this color by clicking on the ellipsis in the properties window. A color palette will appear which allows you to select a color. You can also type the hexadecimal value of a color directly into the settings box.
Side Effects	Any selected rows will automatically be redrawn in the new color.
See Also	<u>Multiple Selection Overview</u>

Grid SelectedForeColor Property

Usage	*grid.***SelectedForeColor** = *colorvalue*
	Read/Write at design time and run time.
	Default is black.
Description	This property denotes the foreground color to be used for rows which have been selected by the user, or have been added to the <u>BookmarkList</u>. Selected rows will only appear if <u>SelectMode</u> is set properly.
	You can set this color by clicking on the ellipsis in the properties window. A color palette will appear which allows you to select a color. You can also type the hexadecimal value of a color directly into the settings box.
Side Effects	Any selected rows will automatically be redrawn in the new color.
See Also	<u>Multiple Selection Overview</u>

Grid SelectMode Property

Usage   *grid.***SelectMode** = *numericexpression*

Read/Write at design time and run time.

Values   0 - Disabled (default)
1 - Enabled
2 - Disabled & Hidden
3 - Enabled & Hidden

Description   SelectMode specifies whether or not the selection is visible, and whether the user may select and deselect rows using the mouse.   By default, selections are visible, but the user may not select rows.

The SelectMode property has 4 possible values:

0 - Disabled.   This is the default.   The user cannot select rows using the mouse.   However, if there is an existing selection present (such as one created by adding to BookmarkList), then it will be displayed.

1 - Enabled.   The user can select rows using the mouse, and any selected rows are displayed.

2 - Disabled & Hidden.   Like Disabled, the user cannot select rows.   In addition, any existing selection will not be visible.   However, the selection is maintained internally.

3 - Enabled & Hidden.   An odd setting, where the user may select rows, but they won't be shown.   If you find a use for this, let us know.

Changing the value of SelectMode at runtime will cause the selection to be hidden or shown if necessary.

See Also   Multiple Selection Overview

Grid SelLength, SelStart, SelText Properties

Usage   *grid.***SelLength** *= numericexpression*
              *grid.***SelStart** *= numericexpression*
              *grid.***SelText** *= stringexpression*

              Read/Write at run time.   Not available at design time.

Description These properties are identical to their counterparts in Visual Basic text controls, and apply to the grid's text editing window.   When the user has selected text for editing, these properties return the length of the selected text, the start of the selection, and the actual text contents of the selection.

              If no cell is currently opened for editing, and only the marquee is present, then SelLength and SelStart will always return zero, and SelText will return an empty string.

              Note that the color of the edit area may have to be changed using EditForeColorand EditBackColor for the selection to become visible.   This is because the default editing color is the same as the highlight color.

Grid Text Property

Usage   *grid.***Text** = *stringexpression*

Read/Write at run time.   Not available at design time.

Description   The Text property may be used to retrieve or set the contents of the current cell being edited by the user.   It has no effect upon the data in the database, and does not cause the <u>Validate</u> or <u>Update</u> events to be sent under any circumstances

If the current cell is currently open for editing by the user, then this property returns the text they are currently working on.   If the Text property is set, then the information in the edited cell will change to match the new value and the old text will be lost.   As stated, Update and Validate will not be triggered.

If the current cell is not being edited (the marquee is displayed around the current cell), then this property will return the visible contents of the current cell.   This may *not* be the value stored in the database, since the grid does not refresh its display until it receives control during the idle loop.   Thus, the following sequence of steps will not work as expected:

```
' Open a file and attempt to fetch cell data

Data1.DatabaseName = "BIBLIO.MDB" ' grid using ds=Data1
Data1.RecordSource = "Titles"
Data1.Refresh
Grid1.RowIndex = 1
Grid1.ColumnIndex = 1
dataval$ = Grid1.Text' will not work
```

If you assign to the Text property while the current cell is not being edited, then the grid will enter edit mode immediately and set the current cell data to the new value of the Text property.   Thus, the following line of code will cause the grid to enter editing mode immediately:

```
Grid1.Text = Grid1.Text
```

If the current cell is not visible when the assignment occurs, the grid will be scrolled so that the cell is visible before edit mode is entered.

Attempting to manipulate the current cell has its limitations.   The grid will leave editing mode automatically under many circumstances (such as when focus is lost).   Therefore, code which manipulates the current cell being edited shouldn't be used in contexts where focus might be lost easily (such as in another button on the form).

Side Effects   When set, causes the current cell to enter edit mode and changes the edited text to the new text.   No side effects upon access.

Grid TopRow Property

Usage   *grid.***TopRow** = *numericexpression*

Read/Write at run time.   Not available at design time.

Description   This property denotes the one-based index of the topmost grid row.   You can use this property to scroll the grid vertically in code at run time just as the RowIndex property can be used.

To the grid, the term row number refers to the row within the overall set of data being browsed, rather than the screen row number.   Whether in callback or database mode, row 1 does not refer to the topmost row of the displayed grid, but refers to the first row of data.   A grid may contain thousands or even millions of rows.

In database mode, it is possible to set TopRow to a value larger than the actual number of rows in the database.   See the description of Rows and RowIndex for a discussion of how database mode treats the row number and row count.

Side Effects   The grid will be redisplayed so that the top row is the selected row.   Upon access, there are no side effects.

Grid VertColor Property

Usage   *grid.***VertColor** = *colorvalue*

Read/Write at design time and run time.
Default is System Window Frame Color (0x80000006)

Description   This property denotes the color of the grid's vertical lines.   This color is only meaningful if vertical lines are turned on, via the VertLines property.

You can set this color by clicking on the ellipsis in the properties window.   A color palette will appear which allows you to select a color.   You can also type the hexadecimal value of a color directly into the settings box.

Side Effects   The vertical lines will automatically be redrawn in the new color.

Grid VertLines Property

Usage	*grid.***VertLines** = *numericexpression*
	Read/Write at design time and run time.
Values	0 - None (default) 1 - Single 2 - 3D
Description	This property determines whether the grid displays vertical lines between rows.
	If the value of this property is 0, then no lines are displayed. If the value is 1, then single-width lines are displayed in the color specified by <u>VertColor</u>.
	If the value of this property is 2, then the grid will attempt to draw the lines in 3D mode. The grid uses the VertColor value for the darker border of the 3D "groove" between cells, and chooses another color from the available colors in the system palette. Some experimentation is usually required to achieve the desired effect. The standard grey 3D effect is chosen by selecting a dark grey color for the lines and a medium grey for the grid background color.
Side Effects	The grid will be redrawn in its entirety to effect the change.

Grid Standard Properties

Standard Properties

The grid also supports the following standard Visual Basic properties.   Consult the Visual Basic documentation for details.

BackColor	BorderStyle	DragIcon	DragMode
Enabled	FontBold	FontItalic	FontName
FontSize	FontStrikeThru	FontUnderLine	ForeColor
Height	HelpContextId	Index	Left
MousePointer	Name	TabIndex	TabStop
Tag	Top	Visible	Width

Grid Append Event

Syntax   Grid_Append()

Description   This event is sent whenever the user is at the bottom row of data in the grid, and they attempt to move down one more row by using the down arrow key.

This event can be used to implement dBASE-like append behavior, where a record is automatically added if you are at the last row of the grid and you attempt to move down further.   Be careful, however, in implementing this.   If the user performs this operation unintentionally, then you will have to delete the newly added record.

If this event is not implemented, then the grid takes no action whatsoever if an attempt is made to move beyond the last row of data.   The marquee or editing area remains at the same location.

Grid CellChange Event

Syntax   Grid_CellChange()

Description   This event is sent whenever the current cell position is changed.   The event is sent under several circumstances:

1. When the grid is first displayed.

2. When the user moves the cell pointer by using the mouse or the keyboard.

3. When data in the grid is changed and causes the cell to change implicitly (such as when the last record in a set is deleted, and the cell was currently positioned on the same row).

4. When the RowIndex or ColumnIndex properties are set to a different value than they currently have.

Typically, the RowIndex and ColumnIndex properties are used to determine the new cell number.   In some cases, this event is triggered with a RowIndex or ColumnIndex property value of zero.   These circumstances occur when there are no rows in the grid at all.

When multiple events are sent, the order will be Change, RowChange, ColumnChange, and finally CellChange.   None of these events will be sent until the data has been validated using the Validate event.

Grid Change Event

Syntax   Grid_Change()

Description   This event is triggered each time the user changes text in the text editing window.   Just as with the Visual Basic Text control, each keystroke causes a Change event to be triggered.

This event is never sent when editing is not in progress.

When multiple events are sent, the order will be Change, RowChange, ColumnChange, and finally CellChange.   None of these events will be sent until the data has been validated using the Validate event.

Grid ColumnChange Event

Syntax　　　　Grid_ColumnChange()

Description　　This event is sent whenever the current column number is changed.　The event is sent under several circumstances:

1. When the grid is first displayed.

2. When the user moves to a new column by using the mouse or the keyboard.

3. When the ColumnIndex property is changed to a different value than it currently has, or the ColumnVisible property for the current column is set to FALSE.

4. If the current column size is decreased so that it no longer appears at all.

Typically, the RowIndex and ColumnIndex properties are used to determine the new cell number.　In some cases, this event is triggered with a RowIndex or ColumnIndex property value of zero.　These circumstances occur when there are no rows in the grid at all.

When multiple events are sent, the order will be Change, RowChange, ColumnChange, and finally CellChange.　None of these events will be sent until the data has been validated using the Validate event.

Grid Fetch Event

Syntax   Grid_Fetch(Row As Long, Col As Integer, Value As String)

Description In callback mode, this event is sent whenever the grid needs to display the contents of a cell.   Just set the Value parameter to the desired string value.

In database mode, this event is not sent.

Grid RowChange Event

Syntax   Grid_RowChange()

Description   This event is sent whenever the current row position is changed.   The event is sent under several circumstances:

1. When the grid is first displayed.

2. When the user moves the cell pointer to another row by using the mouse or the keyboard.

3. When data in the grid is changed and causes the row to change implicitly (such as when the last record in a set is deleted, and the cell was currently positioned on the same row).

4. When the RowIndex property is changed to a different value than it currently has.

Typically, the RowIndex and ColumnIndex properties are used to determine the new cell number.   In some cases, this event is triggered with a RowIndex or ColumnIndex property value of zero.   These circumstances occur when there are no rows in the grid at all.

To TrueGrid, the value of RowIndex refers to the row within the overall set of data being browsed, rather than the screen row number.   Whether in callback or database mode, row 1 does not refer to the topmost row of the displayed grid, but refers to the first row of data.   A grid may contain thousands or even millions of rows.

This event should be used to determine when the user selects a new record.   Although the Click event can also be used, it is not as reliable, since it doesn't allow for keyboard activity.

When multiple events are sent, the order will be Change, RowChange, ColumnChange, and finally CellChange.   None of these events will be sent until the data has been validated using the Validate event.

Grid Update Event

Syntax   Grid_Update(Row As Long, Col As Integer, Value As String)

Description   This event is sent whenever the user has changed the text in the specified cell and is about to commit the change by moving to a different cell.   If this event is sent, then the contents of the cell have already been validated.

This event is most useful in callback mode, where it is the only way changes to data in the grid can take effect.   Typically, callback mode users will use this event to store the grid data someplace where it can be accessed the next time a <u>Fetch</u> event is sent.

In database mode, all data storage is automatic.   However, actual record data is not updated by Visual Basic until the row position changes.   Thus, each cell update reflects that the *user* has changed the data and it is being held in the internal grid change buffer until it is written to the database.

Grid Validate Event

Event   Validate

Syntax   Grid_Validate(Row As Long, Col As Integer, Value As String, Cancel As Integer)

Description   In both database and callback modes, this event is sent whenever the user has changed the text in the specified cell and is about to commit the change by moving to a different cell.   If this event is not used, all changes are valid.   If this event code executes without setting the value of the Cancel parameter, then the specified value is valid.

Setting the Cancel parameter to *True* causes the grid to remain in editing mode until the data is valid.   If the Validate handler is triggered because the grid has lost focus, then setting Cancel to *True* will cause the grid to force focus to return to the editing area automatically.

The validation code can perform any steps necessary to validate the input data. However, the code should avoid accessing or changing the current database (in database mode), or setting any properties which may affect the current cell position in the grid.

Note that actual record data is not written to the database until the current row position changes.   Thus, the Validate handler can be used to validate data immediately upon entry in the grid.   Additional validation may be necessary (or desirable) in the data control Validate handler, since it is called when the actual database update is about to occur.

Grid Standard Events

Standard Events

The grid also supports the following standard Visual Basic events.   Consult the Visual Basic documentation for details.

Click	DblClick	DragDrop	DragOver
GotFocus	KeyDown	KeyPress	KeyUp
LostFocus	MouseDown	MouseMove	MouseUp

Visual Basic Format Strings

TrueGrid uses standard Visual Basic format strings to provide display editing for columns within the Agility grid control.   In Visual Basic, the **Format$** function is used to edit values.   These format strings may be entered at design time using the layout editor "Edit Mask..." option or by using the <u>EditMask</u> property. The following paragraphs describe the Visual Basic format string capabilities.

To format numbers, you can use the commonly-used formats that have been predefined in Visual Basic or you can create user-defined formats with standard characters that have special meaning when used in a format expression.

The following table shows the predefined numeric format names you can use and the meaning of each:

Format name	Description
General Number	Display the number as is, with no thousand separators.
Currency	Display number with thousand separator, if appropriate; display negative numbers enclosed in parentheses; display two digits to the right of the decimal separator.
Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Standard	Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator.
Percent	Display number multiplied by 100 with a percent sign (%) appended to the right; display two digits to the right of the decimal separator.
Scientific	Use standard scientific notation.
Yes/No	Display No if number is 0, otherwise display Yes.
True/False	Display False if number is 0, otherwise display True.
On/Off	Display Off if number is 0, otherwise display On.

The following table shows the characters you can use to create user-defined number formats and the meaning of each:

Character	Meaning
Null string	Display the number with no formatting.
0	Digit placeholder.
	Display a digit or a zero. If there is a digit in the expression being formatted in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position.
	If the number being formatted has fewer digits than there are zeros (on either side of the decimal) in the format expression, leading or trailing zeros are displayed. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, the number is rounded to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, the extra digits are displayed without modification.
#	Digit placeholder.
	Display a digit or nothing. If there is a digit in the expression being formatted in the position where the # appears in the format string, display it; otherwise, display nothing in that position.
	This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has the same or fewer digits than there are # characters on either side of the decimal separator in the format expression.
.	Decimal placeholder.
	The decimal placeholder determines how many digits are displayed to the left and

right of the decimal separator.   If the format expression contains only number signs to the left of this symbol, numbers smaller than 1 begin with a decimal separator.   If you want a leading zero to always be displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator instead.   The actual character used as a decimal placeholder in the formatted output depends on the Number Format specified in the International section of the Microsoft Windows Control Panel.   For some countries, a comma is used as the decimal separator.

| % | Percentage placeholder. |

The expression is multiplied by 100.   The percent character (%) is inserted in the position where it appears in the format string.

| , | Thousand separator. |

The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator.   Standard use of the thousand separator is specified if the format contains a comma surrounded by digit placeholders (0 or #).   Two adjacent commas or a comma immediately to the left of the decimal separator (whether or not a decimal is specified) means "scale the number by dividing it by 1000, rounding as needed."   You can scale large numbers using this technique.   For example, you can use the format string "##0,," to represent 100 million as 100.   Numbers smaller than 1 million are displayed as 0.   Two adjacent commas in any position other than immediately to the left of the decimal separator are treated simply as specifying the use of a thousand separator.   The actual character used as the thousand separator in the formatted output depends on the Number Format specified in the International section of the Control Panel.   For some countries, a period is used as the thousand separator.

| E- E+ e- e+ | Scientific format. |

If the format expression contains at least one digit placeholder (0 or #) to the right of E-, E+, e-, or e+, the number is displayed in scientific format and E or e is inserted between the number and its exponent.   The number of digit placeholders to the right determines the number of digits in the exponent.   Use E- or e- to place a minus sign next to negative exponents.   Use E+ or e+ to place a minus sign next to negative exponents and a plus sign next to positive exponents.

| : | Time separator. |

The time separator separates hours, minutes, and seconds when time values are formatted.   The actual character used as the time separator depends on the Time Format specified in the International section of the Control Panel.

| / | Date separator. |

The date separator separates the day, month, and year when date values are formatted.   The actual character used as the date separator in the formatted output depends on Date Format specified in the International section of the Control Panel.

| - + $ ( ) space | Display a literal character. |

To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks (" ").

| \ | Display the next character in the format string. |

Many characters in the format expression have a special meaning and can't be displayed as literal characters unless they are preceded by a backslash.   The backslash itself isn't displayed.   Using a backslash is the same as enclosing the next character in double quotation marks.   To display a backslash, use two backslashes (\\).

Examples of characters that can't be displayed as literal characters are the date- and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y, and /:), the numeric-formatting characters (#, 0, %, E, e, comma, and period), and the string-formatting characters (@, &, <, >, and !)."ABC" Display the string inside the double quotation

marks.

To include a string in fmt from within Visual Basic, you must use Chr(34) to enclose the text (34 is the ANSI code for a double quotation mark).

*   Display the next character as the fill character.

Any empty space in a field is filled with the character following the asterisk.   Unless the fmt argument contains one of the predefined formats, a format expression for numbers can have from one to four sections separated by semicolons.

If you use	The result is
One section only	The format expression applies to all values.
Two sections	The first section applies to positive values and zeros, the second to negative values.
Three sections	The first section applies to positive values, the second to negative values, and the third to zeros.
Four sections	The first section applies to positive values, the second to negative values, the third to zeros, and the fourth to Null values.

The following example has two sections: the first defines the format for positive values and zeros; the second section defines the format for negative values.

     "$#,##0;($#,##0)"

If you include semicolons with nothing between them, the missing section is printed using the format of the positive value.   For example, the following format displays positive and negative values using the format in the first section and displays "Zero" if the value is zero.

     "$#,##0;;\Z\e\r\o"

Some sample format expressions for numbers are shown below. (These examples all assume that Country is set to United States in the International section of the Control Panel.)   The first column contains the format strings.   The other columns contain the output that results if the formatted data has the value given in the column headings.

Format (fmt)	Positive 5	Negative 5	Decimal .5	Null
Null string	5	-5	0.5	
0	5	-5	1	
0.00	5.00	-5.00	0.50	
#,##0	5	-5	1	
#,##0.00;;;Nil	5.00	-5.00	0.50	Nil
$#,##0;($#,##0)	$5	($5)	$1	
$#,##0.00;($#,##0.00)	$5.00	($5.00)	$0.50	
0%	500%	-500%	50%	
0.00%	500.00%	-500.00%	50.00%	
0.00E+00	5.00E+00	-5.00E+00	5.00E-01	
0.00E-00	5.00E00	-5.00E00	5.00E-01	

Numbers can also be used to represent date and time information.   You can format date and time serial numbers using date and time formats or number formats because date/time serial numbers are stored as floating-point values.

To format dates and times, you can use either the commonly used formats that have been predefined in Visual Basic or create user-defined time formats using standard characters that have special meaning when used in a format expression.

The following table shows the predefined data format names you can use and the meaning of each:

Format Name	Description
General Date	Display a date and/or time. For real numbers, display a date and time. (e.g. 4/3/93

05:34 PM); If there is no fractional part, display only a date (e.g. 4/3/93); if there is no integer part, display time only (e.g. 05:34 PM).

Long Date	Display a Long Date, as defined in the International section of the Control Panel.
Medium Date	Display a date in the same form as the Short Date, as defined in the International section of the Control Panel, except spell out the month abbreviation.
Short Date	Display a Short Date, as defined in the International section of the Control Panel.
Long Time	Display a Long Time, as defined in the International section of the Control Panel. Long Time includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minutes and the AM/PM designator.
Short Time	Display a time using the 24-hour format (e.g. 17:45)

The following table shows the characters you can use to create user-defined date/time formats and the meaning of each:

Character	Meaning
c	Display the date as ddddd and display the time as t t t t t, in that order. Only date information is displayed if there is no fractional part to the date serial number; only time information is displayed if there is no integer portion.
d	Display the day as a number without a leading zero (1-31).
dd	Display the day as a number with a leading zero (01-31).
ddd	Display the day as an abbreviation (Sun-Sat).
dddd	Display the day as a full name (Sunday-Saturday).
ddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the Short Date setting in the International section of the Windows Control Panel. The default Short Date format is m/d/yy.
dddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the Long Date setting in the International section of the Control Panel. The default Long Date format is mmmm dd, yyyy.
w	Display the day of the week as a number (1 for Sunday through 7 for Saturday.)
ww	Display the week of the year as a number (1-53).
m	Display the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan-Dec).
mmmm	Display the month as a full month name (January-December).
q	Display the quarter of the year as a number (1-4).
y	Display the day of the year as a number (1-366).
yy	Display the year as a two-digit number (00-99).
yyyy	Display the year as a four-digit number (100-9999).
h	Display the hour as a number without leading zeros (0-23).
hh	Display the hour as a number with leading zeros (00-23).
n	Display the minute as a number without leading zeros (0-59).
nn	Display the minute as a number with leading zeros (00-59).
s	Display the second as a number without leading zeros (0-59).
ss	Display the second as a number with leading zeros (00-59).
t t t t t	Display a time serial number as a complete time (including hour, minute, and second) formatted using the time separator defined by theTime Format in the International section of the Control Panel. A leading zero is displayed if the Leading Zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is

h:mm:ss.

AM/PM	Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour between noon and 11:59 PM.
am/pm	Use the 12-hour clock and display a lowercase AM with any hour before noon; display a lowercase PM with any hour between noon and 11:59 PM.
A/P	Use the 12-hour clock and display an uppercase A with any hour before noon; display an uppercase P with any hour between noon and 11:59 PM.
a/p	Use the 12-hour clock and display a lowercase A with any hour before noon; display a lowercase P with any hour between noon and 11:59 PM.
AMPM	Use the 12-hour clock and display the contents of the 1159 string (s1159) in the WIN.INI file with any hour before noon; display the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as it exists in the WIN.INI file. The default format is AM/PM.

The following are examples of user-defined date and time formats:

Format	Display
m/d/yy	12/7/58
d-mmmm-yy	7-December-58
d-mmmm	7 December
mmmm-yy	December 58
hh:mm AM/PM	08:50 PM
h:mm:ss a/p	8:50:35 p
h:mm	20:50
h:mm:ss	20:50:35
m/d/yy h:mm	12/7/58 20:50

Strings can also be formatted with Format[$].   A format expression for strings can have one section or two sections separated by a semicolon.

If you use	The result is
One section only	The format applies to all string data.
Two sections	The first section applies to string data, the second to Null values and zero-length strings.

You can use any of the following characters to create a format expression for strings:

Character	Meaning
@	Character placeholder.
	Display a character or a space. If there is a character in the string being formatted in the position where the @ appears in the format string, display it; otherwise, display a space in that position. Placeholders are filled from right to left unless there is an ! character in the format string. See below.
&	Character placeholder.
	Display a character or nothing. If there is a character in the string being formatted in the position where the & appears, display it; otherwise, display nothing. Placeholders are filled from right to left unless there is an ! character in the format string. See below.
<	Force lowercase.
	All characters are displayed in lowercase format.
>	Force uppercase.
	All characters are displayed in uppercase format.

!   Force placeholders to fill from left to right instead of right to left.