
ComponentOne

True DBGrid for WinForms

ComponentOne, a division of GrapeCity

201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Website: <http://www.componentone.com>

Sales: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of GrapeCity, Inc. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for \$25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

Table of Contents

| | |
|--|-------|
| True DBGrid for WinForms | 10 |
| Help with WinForms Edition | 10 |
| Differences Between True DBGrid for WinForms and FlexGrid for WinForms | 10 |
| Key Features | 11-12 |
| True DBGrid for WinForms Quick Start | 13 |
| Step 1 of 3: Creating a True DBGrid for WinForms Application | 13 |
| Step 2 of 3: Binding True DBGrid for WinForms to a DataSet | 13-16 |
| Step 3 of 3: Customizing True DBGrid for WinForms Settings | 16-18 |
| True DBGrid for WinForms Top Tips | 19-20 |
| Object Model | 21 |
| True DBGrid for WinForms Objects and Collections | 21-23 |
| C1TrueDBGrid Class | 23 |
| C1TrueDBDropDown Class | 23 |
| C1DataColumnCollection Class | 23 |
| C1DataColumn Object | 23 |
| C1DisplayColumnCollection Class | 23-24 |
| C1DisplayColumn Class | 24 |
| GroupedColumnCollection Class | 24 |
| SplitCollection Class | 24 |
| Split Object | 24-25 |
| GridStyleCollection Class | 25 |
| Style Object | 25 |
| ValueItems Class | 25-26 |
| ValueItemCollection Class | 26 |
| ValueItem Class | 26 |
| PrintInfo Class | 26 |
| PrintPreviewWinSettings Class | 26 |
| HBar Class | 26-27 |
| VBar Class | 27 |
| GridLines Class | 27 |
| GridBorders Class | 27 |
| SelectedRowCollection Class | 27 |
| SelectedColumnCollection Class | 27 |
| Working with Objects and Collections | 27 |

| | |
|--|-------|
| Working with Collections | 27-32 |
| Adding Members | 32-33 |
| Removing Members | 33 |
| Working with the Count Property | 33-35 |
| Design-Time Support | 36 |
| Understanding the Object Model and Property Access | 36 |
| Accessing Global Grid Properties | 36 |
| Accessing Split-Specific Properties | 36-37 |
| Accessing Column Properties | 37 |
| Using the Split Collection Editor | 37-38 |
| Splits Properties | 38-40 |
| Using the C1DisplayColumnCollection Editor | 40-41 |
| DisplayColumns Properties | 41-42 |
| Using the ValueItemCollection Editor | 42-43 |
| Using the C1TrueDBGrid Style Editor | 43-44 |
| Using the C1TrueDBGrid Designer | 44 |
| Accessing the C1TrueDBGrid Designer | 44-45 |
| C1TrueDBGrid Designer Elements | 45-47 |
| Splits Properties | 47-49 |
| C1DataColumn Properties | 49-50 |
| DisplayColumns Properties | 50-51 |
| C1TrueDBGrid Tasks Menu | 51-58 |
| Column Tasks Menu | 58-65 |
| C1TrueDBGrid Context Menu | 65-67 |
| Run-Time Interaction | 68 |
| Navigation and Scrolling | 68 |
| Mouse Interaction | 68 |
| Clicking the Rightmost Column | 68-69 |
| Keyboard Interaction | 69-70 |
| Navigation at Row Boundaries | 70 |
| Navigation at Split Boundaries | 70-71 |
| Restricting Cell Navigation | 71-72 |
| Selection, Sorting, and Movement | 72 |
| Selecting Columns | 72-73 |
| Moving Columns | 73-74 |
| Moving Columns at Run Time | 74-75 |

| | |
|---|---------|
| Sorting Columns | 75-76 |
| Selecting Rows | 76 |
| Selecting a Range of Cells | 76 |
| Sizing and Splitting | 76 |
| Sizing Rows | 76-77 |
| Sizing Columns | 77-78 |
| Database Operations | 78 |
| Editing Data | 78-79 |
| Adding a New Record | 79 |
| Deleting a Record | 79 |
| Customized Grid Editors | 79-80 |
| Using Custom Editors | 80-81 |
| Creating Custom Editors | 81 |
| Additional User Interaction Features | 81-82 |
| Data Binding | 83 |
| Binding True DBGrid for WinForms to a Data Source | 83 |
| Preserving the Grid's Layout | 83-84 |
| Using Unbound Columns | 84 |
| Creating Unbound Columns | 84-85 |
| Implementing Multiple Unbound Columns | 85-86 |
| Updating Unbound Columns | 86-87 |
| Editing Unbound Columns | 87-88 |
| Creating an Unbound Grid | 88-89 |
| Adding New Rows to an Unbound Grid | 89-93 |
| Customizing the Grid's Appearance | 94 |
| Visual Styles | 94-96 |
| Captions, Headers, and Footers | 96-97 |
| Column and Grid Captions | 97 |
| Column Footers | 97-98 |
| Multiple-Line Headers and Footers | 98 |
| Split Captions | 98-99 |
| Three-Dimensional vs. Flat Display | 99-101 |
| Borders and Dividing Lines | 101-102 |
| Unpopulated Regions | 102 |
| The Rightmost Column | 102-103 |
| Unused Data Rows | 103-104 |

| | |
|--|---------|
| Highlighting the Current Row or Cell | 104-107 |
| Row Height and Word Wrap | 107 |
| Adjusting the Height of All Grid Rows | 107 |
| Enabling Wordwrap in Cells | 107-108 |
| Alternating Row Colors | 108 |
| Horizontal and Vertical Alignment | 108-109 |
| Data Presentation Techniques | 110 |
| Text Formatting | 110 |
| Numeric Field Formatting | 110 |
| Predefined Numeric Options | 110-111 |
| Custom Number Formatting | 111-112 |
| Input Validation with Built-In Formatting | 112 |
| Formatting with an Input Mask | 112-113 |
| Formatting with a Custom Event Handler | 113-115 |
| Automatic Data Translation with Valueltems | 115 |
| What are Valueltems? | 115 |
| Specifying Text-to-Text Translations | 115-117 |
| Specifying Text-to-Picture Translations | 117-119 |
| Displaying Both Text and Pictures in a Cell | 119-123 |
| Displaying Boolean Values as Check Boxes | 123 |
| Displaying Allowable Values as Radio Buttons | 123-124 |
| Context-Sensitive Help with CellTips | 124-126 |
| Scroll Tracking and ScrollTips | 126-127 |
| Data-Sensitive Cell Merging | 127-132 |
| Formatting Merged Cells | 132-133 |
| Column Grouping | 133-134 |
| Column Grouping with the GroupIntervalEnum Enumeration | 134 |
| Group Rows by Year | 134-137 |
| Group Rows by the First Character of the Value | 137-140 |
| Group Rows by Date Value (Outlook-Style) | 140-144 |
| Group Rows by Custom Setting | 144-149 |
| Expanding and Collapsing Grouped Rows | 149-152 |
| Data Display | 152 |
| Hierarchical Data Display | 152-154 |
| Drop-Down Hierarchical Data Display | 154-155 |

| | |
|---|---------|
| Form Data Display | 155 |
| Inverted Data Display | 155-156 |
| Multiple Line Data Display | 156-157 |
| Implications of Multiple-Line Mode | 157 |
| Multiple Line Fixed Data Display | 157-158 |
| Owner-Drawn Cells | 158-162 |
| Filtering Data in DataSets | 162 |
| Manually Filtering Data | 163 |
| Adding a Watermark to the Filter Bar | 163-165 |
| Filtering the Grid with Multiple Criteria | 165-166 |
| Adding a Filter Drop-Down List | 166-167 |
| Condition Filtering | 167-168 |
| Custom Filtering | 168-169 |
| How to Use Splits | 170 |
| Referencing Splits and their Properties | 170-172 |
| Split Properties Common to C1TrueDBGrid | 172 |
| Split-Only Properties Not Supported by C1TrueDBGrid | 172-173 |
| Split Matrix Notation | 173 |
| Creating and Removing Splits | 173-175 |
| Working with Columns in Splits | 175 |
| Sizing and Scaling Splits | 175-179 |
| Creating and Resizing Splits through User Interaction | 179-181 |
| Vertical Scrolling and Split Groups | 181-183 |
| Horizontal Scrolling and Fixed Columns | 184-185 |
| Navigation Across Splits | 185 |
| How to Use Styles | 186 |
| Built-In Named Styles | 186-187 |
| Named Style Defaults | 187-188 |
| Named Style Inheritance | 188-189 |
| Modifying Named Styles | 189-190 |
| Working with Style Properties | 190 |
| Modifying a Style Property Directly | 190-191 |
| Named Styles vs. Anonymous Styles | 191-192 |
| Anonymous Style Inheritance | 192 |
| Example 1 of 10: Inheriting from Containing Splits | 192-193 |
| Example 2 of 10: Affecting Only Data Cells in the First Split | 193-194 |

| | |
|--|---------|
| Example 3 of 10: Affecting All Elements Only in the First Split | 194-195 |
| Example 4 of 10: Affecting Only Data Cells in the First Column of the First Split | 195 |
| Example 5 of 10: Affecting All Elements Only in the First Column of the First Split | 195-196 |
| Example 6 of 10: Changing the BackColor of the Style Property | 196-197 |
| Example 7 of 10: Changing Only the Data Cells in the First Split | 197 |
| Example 8 of 10: Changing Only the Data Cells in the First Column of the First Split | 197-198 |
| Example 9 of 10: Setting the Alignment of C1DisplayColumn Objects | 198-199 |
| Example 10 of 10: Setting the Alignment for Column Headers | 199 |
| Applying Styles to Cells | 199-200 |
| Specifying Cell Status Values | 200-201 |
| Applying Cell Styles by Status | 201-203 |
| Applying Cell Styles by Contents | 203-204 |
| Applying Cell Styles by Custom Criteria | 204-207 |
| Cell Style Evaluation Order | 207 |
| Applying Pictures to Grid Elements | 207-208 |
| Displaying Background Pictures | 208-212 |
| Displaying Foreground Pictures | 212-213 |
| Cell Editing Techniques | 214 |
| How Cell Editing Works | 214 |
| Initiating Cell Editing | 214 |
| Color and Wordwrap | 214 |
| Determining Modification Status | 214 |
| Determining Cell Contents | 214-215 |
| Terminating Cell Editing | 215 |
| Handling Editing Events | 215 |
| Standard Keystroke Events | 215-216 |
| Column Editing Events | 216-217 |
| Changing Cell Contents with a Single Keystroke | 217-219 |
| Working with Text | 219 |
| Limiting the Size of Data Entry Fields | 219 |
| Providing a Drop-Down Edit Control for Long Fields | 219-220 |
| Selecting and Replacing Text | 220 |
| Input Masking | 220 |
| Specifying an Input Mask for a Column | 220-221 |
| Using an Input Mask for Formatting | 221 |
| Controlling How Masked Input is Updated | 221-222 |

| | |
|---|---------|
| In-Cell Buttons | 222 |
| Enabling the In-Cell Button | 222-223 |
| Rendering Cells as Command Buttons | 223-224 |
| Detecting In-Cell Button Clicks | 224 |
| Customizing the In-Cell Button Bitmap | 224-225 |
| Drop-Down Controls | 225 |
| Using the Built-In Combo Box | 225 |
| Detecting Built-In Combo Box Selections | 225-226 |
| Using the C1TrueDBDropDown Control | 226 |
| Automatic Data Translation with C1TrueDBDropDown | 226-227 |
| Using an Arbitrary Drop-Down Control | 227-228 |
| Using the Built-In Column Button | 228 |
| True DBGrid for WinForms Samples | 229-230 |
| True DBGrid for WinForms Tutorials | 231 |
| Tutorial 1: Binding True DBGrid to a DataSet | 231-234 |
| Tutorial 2: Using True DBGrid for WinForms with SQL Query Results | 234-237 |
| Tutorial 3: Linking Multiple True DBGrid Controls | 237-240 |
| Tutorial 4: Interacting with Code and Other Bound Controls | 240-246 |
| Tutorial 5: Selecting Multiple Rows Using Bookmarks | 246-248 |
| Tutorial 6: Defining Unbound Columns in a Bound Grid | 248-250 |
| Tutorial 7: Displaying Translated Data with the Built-In Combo | 250-252 |
| Tutorial 8: Attaching a Drop-Down Control to a Grid Cell | 252-253 |
| Tutorial 9: Attaching an Arbitrary Drop-Down Control to a Grid Cell | 253-258 |
| Tutorial 10: Enhancing the User Interface with In-Cell Bitmaps | 258-260 |
| Tutorial 11: Using Styles to Highlight Related Data | 260-263 |
| Tutorial 12: Displaying Rows in Alternating Colors | 263-265 |
| Tutorial 13: Implementing Drag-and-Drop Functionality | 265-271 |
| Tutorial 14: Creating a Grid with Fixed, Nonscrolling Columns | 271-273 |
| Tutorial 15: Using PrintInfo and Print Preview | 273-276 |
| Tutorial 16: Using the Hierarchical Display | 276-277 |
| Tutorial 17: Creating a Grouping Display | 277-278 |
| Tutorial 18: Using Value Translation | 278-279 |
| Tutorial 19: Using Range Selection | 279-281 |
| Tutorial 20: Displaying Multiple Data Views | 281-285 |
| Tutorial 21: Adding a Filter Bar | 285-286 |

| | |
|--|---------|
| Tutorial 22: Borders, Scroll Tracking, and Scroll Tips | 286-295 |
| True DBGrid for WinForms Task-Based Help | 296 |
| Adding a New Row to C1TrueDBGrid | 296-298 |
| Selecting a Row | 298-299 |
| Accessing the Values of the Selected Rows in the Grid | 299-301 |
| Controlling Grid Interaction | 301 |
| Disabling Column Sorting | 301-302 |
| Locking a Cell from Being Edited | 302-303 |
| Freezing Columns | 303-304 |
| Restricting Editing in Specific Columns | 304-306 |
| Setting the Grid's Appearance | 306 |
| Adding a Gradient Fill to a Column | 306-310 |
| Formatting Rows by Specific Criteria | 310-313 |
| Hiding the Record Selectors Column | 313-314 |
| Highlighting the Row of the Selected Cell | 314-317 |
| Disabling Selected Highlight | 317-318 |
| Placing an Image in a Column Header | 318-322 |
| Setting Multiple Height Values for Rows | 322-324 |
| Setting the Background Color of a Row | 324-325 |
| Setting the Column's Caption Height | 325-327 |
| Setting the Font Style of a Column | 327-331 |
| Aligning the Column Headers | 331-332 |
| Moving the Focus in Code | 332-333 |
| Adding Custom Error Checking to C1TrueDBGrid | 333-334 |
| Changing the Column Order in the Grid | 334-336 |
| Resizing Columns During Grid Resizing | 336-337 |
| Exporting Grid Data | 337-338 |
| Exporting To All Available File Types | 338-339 |
| Exporting to Delimited Text | 339-340 |
| Exporting to Excel | 340-342 |
| Exporting to HTML | 342-343 |
| Exporting to PDF | 343-344 |
| Exporting to RTF | 344-345 |
| Getting the DataRow for a Row Index After Sorting or Filtering | 345 |
| Modifying the ConnectionString | 345-347 |
| Moving to the AddNew Row | 347-349 |

| | |
|--|---------|
| Saving the Layout of the Grid | 349-350 |
| Searching for Entries in a Column | 350-353 |
| Setting Default Values for New Rows | 353-354 |
| Displaying a Column Total in the Footer | 354-356 |
| Displaying the Current Column and Row | 356-357 |
| Displaying the Date and Time in a Column | 357-358 |
| Programmatically Entering Edit Mode | 358-359 |
| Changing the Filter Language | 359-360 |
| Creating a Custom Print Preview | 360-364 |

True DBGrid for WinForms

True DBGrid for WinForms is a set of robust, easy-to-use .NET grid controls that allow you to create complex bound and unbound grid applications quickly. **True DBGrid for WinForms**'s strength is data binding; with an ADO.NET managed database interface, **True DBGrid for WinForms** offers features like Excel-like split views and built-in hierarchical binding and grouping that'll boost end-user productivity. Ease of use extends to the developer, too; controls are based on Microsoft specifications, so you'll have no problems using **True DBGrid for WinForms** if you're familiar with the Microsoft .NET object and collection models. The Target framework for WindowsForms applications which use .Net 4 build of C1TrueDBGrid control should be **.NET Framework 4 Client Profile**.

With two controls, [C1TrueDBGrid](#), a full-featured grid control, and [C1TrueDBDropDown](#), a multicolumn drop-down list box for a grid column, **True DBGrid for WinForms** includes dozens of advanced features including data access, data presentation (such as splits, grouping, filtering, and customized navigation), and user interface features (including Office 2007 and Office 2010 Visual Styles), that you can use to build intuitive, reliable, professional-looking grid applications.



Getting Started

Get started with the following topics:

- [Key Features](#)
- [Quick Start](#)
- [Object Model](#)
- [Samples](#)
- [Tutorials](#)

Help with WinForms Edition

Getting Started

For information on installing **ComponentOne Studio WinForms Edition**, licensing, technical support, namespaces and creating a project with the control, please visit [Getting Started with WinForms Edition](#).

Differences Between True DBGrid for WinForms and FlexGrid for WinForms

Many customers ask about the differences between our WinForms grid components. While both are robust, easy-to-use grid controls that allow you to browse, edit, add, delete, and manipulate tabular data, there are several reasons why you may want to use one over the other.

Both components can be used in bound or unbound mode, but **True DBGrid for WinForms** allows you to work more easily in bound mode. **True DBGrid for WinForms** is better suited for data binding and, therefore, offers more features in that area, including split views and built-in hierarchical binding and grouping.

FlexGrid for WinForms, on the other hand, is more suited for working with unbound data; **FlexGrid for WinForms** includes features that allow you to customize trees and take advantage of its cell merging capabilities. You can also derive from it to create customized grids.


If you plan to work with your grids in unbound mode or need to customize the grid beyond what the object model offers, **FlexGrid for WinForms** is the right choice. However, if you plan to use your grids mainly in bound mode and require advanced features such as splits and hierarchical views, **True DBGrid for WinForms** is the better choice.

If you have additional questions about **True DBGrid for WinForms** and **FlexGrid for WinForms**, please visit our Web site at <http://www.componentone.com>.

Key Features

True DBGrid for WinForms includes dozens of advanced data access, data presentation, and user interface features that enable developers to build intuitive, professional-looking applications:

- **Extensive Design-time Support**
Design-time features, including SmartTags and full-featured editors allow you to intuitively create grid applications with little or no coding. For details, see [Design-Time Support](#).
- **Multiple Data Views**
Present data in the format that's most useful for you with GroupBy View and standard Microsoft Outlook-style grouping, Hierarchical Data Display, Form View, Inverted View, MultipleLines View, and MultipleLinesFixed View. See [Data Display](#) for more information.
- **Horizontal and Vertical Splits**
Excel-like splits let you split the grid horizontally, vertically, or both. Plus, you have control over how splits scroll, individually or together. For details, see [How to Use Splits](#).
- **Drop-Down Object Support**
Include a variety of drop-down objects for data entry, including a multicolumn control (the [C1TrueDBDropDown](#) control), a combo box, and a multiline text editor. See [Drop-Down Controls](#) for more information. Third-party drop-down controls also supported.
- **Multiple Export Options and Robust Print Options**
Export your grid to multiple formats including Delimited Text, Excel (XLS and XLSX), PDF, HTML, RTF, and more! For details see [Exporting Grid Data](#). Control printing fully with features such as zoom, fit in window, stop pagination, and print preview.

 **Note:** C1TrueDBGrid's export and printing features use **Reports for WinForms'** components internally, and you may need to reference **Reports for WinForms'** assemblies (C1.Win.C1Report and C1.C1Report) if you are receiving an error related to the assembly.

- **Office 2007 and 2010 Styling**
True DBGrid for WinForms supports Visual Styles that mimic the styles available in Office 2007 and Office 2010. You can set the Visual Style easily through the [VisualStyle](#) property. For more information about available Visual Styles, see [Visual Styles](#).
- **Universal .NET Data Binding**
True DBGrid for WinForms can bind to any .NET data source with little or no code, allowing you to create a fully-navigational database browser in seconds. See [Data Binding](#) for more information.
- **Designed to Microsoft Specifications**
True DBGrid for WinForms includes .NET objects designed according to Microsoft specifications so if you're familiar with the Microsoft .NET object and collection models, you'll have no problem using **True DBGrid for WinForms**.
- **Enhanced Keyboard Navigation**
With just one property setting, control the relative position of the next cell when end-users press the ENTER key. See [Navigation and Scrolling](#) for information.
- **Rich Scrolling Capabilities**
Easily track the location of the scroll bar in the grid, set the vertical scroll bar thumb to scroll records as moved, and provide an informational pop-up during scrolling. See [Scroll Tracking and ScrollTips](#) and [Tutorial 22: Borders, Scroll Tracking, and Scroll Tips](#) for more information.
- **2D and 3D Cell Display**
Choose two-dimensional, three-dimensional, or a combination of the two to control cell appearance to your specifications. See [Three-Dimensional vs. Flat Display](#) for more information.
- **Style Border Properties**
Customize the appearance, size, color, and type of cell borders. See [Borders and Dividing Lines](#) for more information.
- **Excel and Word-Like Styles**
Use hierarchical style objects to customize the grid's appearance with font, color, picture, and formatting

specifications.

- **Alternating Row Colors**

Add alternating row colors to the grid to enhance the readability of the grid's display. See [Alternating Row Colors](#) for more information.

- **In-Cell Objects**

Add a variety of in-cell objects for data display and editing in the grid, including bitmaps, command buttons, check boxes, and radio buttons. See [In-Cell Buttons](#) for more information.

- **Automatic Data Translation**

Automatically translate database values into alternate text or graphics without coding in the grid. For example, numeric codes can be rendered as words or even bitmaps. See [Automatic Data Translation with ValueItems](#) for details.

- **Data-Sensitive Display**

Apply different styles to individual cells depending upon their contents. For example, show negative numbers in red, or fields containing a particular substring in bold. See [Applying Styles to Cells](#) for more information.

- **Input Masking**

Assign input templates to columns in order to simplify the run-time data entry process and reduce end-user data entry errors. See [Input Masking](#) for details.

- **Filter Bar**

Implement custom end-user operations such as incremental search and record set filtering using the filter bar, a special data entry row below the column headers. See [Filtering Data in DataSets](#) for more information.

- **Unbound Grids and Columns**

Easily create an unbound grid – you can even add unbound columns to a bound grid. See [Using Unbound Columns](#) and [Creating an Unbound Grid](#) for details.

- **Run-Time CellTips**

Add context-sensitive help for end-users by using cell tips in the grid. See [Context-Sensitive Help with CellTips](#) for details.

- **Fixed, Nonscrolling Columns**

Create fixed, nonscrolling columns anywhere in the grid – create a fixed left or right-most column or even to fix a column in the middle of the grid. Creating fixed columns is also easy to do with splits. See [Freezing Columns](#) for more information.

- **Excel-style Cell Selection**

Choose not only any row or column, but also any range of cells. See [Selection, Sorting, and Movement](#) for more information.

- **Automatic Column Sizing**

Keep your data viewable by resizing columns proportionately whenever the grid is resized horizontally. See [Sizing and Splitting](#) for details.

- **Merge Contiguous Like-valued Cells**

Merge adjacent rows of like-valued data from a specified column into a noneditable cell, or display all cell values individually. See [Data-Sensitive Cell Merging](#) for more information.

- **Simplify Data Entry**

Reduce the number of keystrokes needed for drop-down selection with the **AutoDropdown** and **AutoCompletion** properties to simplify end-user data entry.

- **Extensive Object Model**

The **True DBGrid** has two separate column objects to help simplify the sometimes daunting object model. The [C1DataColumn](#) object contains all of the properties related to data and data processing, while the [C1DisplayColumn](#) object contains all of the properties related to the column's display. See [Object Model](#) for more information.

- **And Much More...**

Customizable ENTER key behavior, drop-down hierarchical grid, tag property for column objects, right to left support, and a wide variety of print enhancements.


True DBGrid for WinForms Quick Start

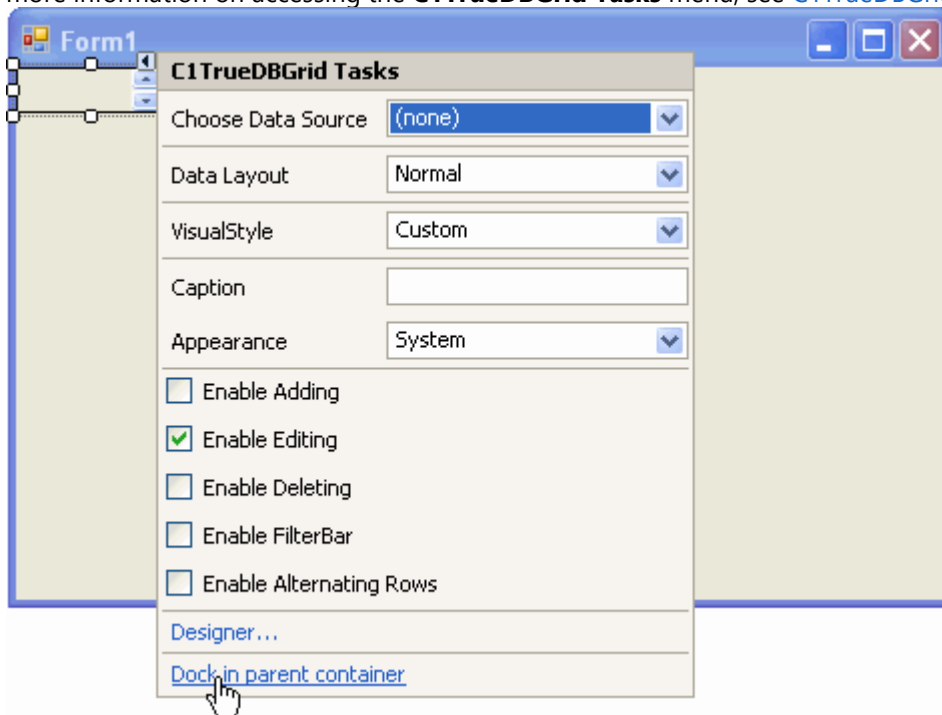
This quick start guide will walk you through the steps of creating a **True DBGrid for WinForms** application, binding the grid to a data source, and customizing the grid's appearance and behavior settings. You'll discover that you can easily create powerful database applications using **True DBGrid for WinForms**.

The quick start uses an Access database, **C1NWind.mdb**. The C1NWind.mdb database file is located in the **Common** subdirectory of the **WinForms Edition** program. The tutorials assume that the database file C1NWind.mdb is in the **Documents\ComponentOne Samples\Common** directory, and refer to it by filename instead of the full pathname for the sake of brevity.

Step 1 of 3: Creating a True DBGrid for WinForms Application

In this step you will add a **C1TrueDBGrid** control to the form and create a simple grid application. Complete the following steps:

1. Create a new .NET project.
2. Open the Visual Studio Toolbox and double-click the **C1TrueDBGrid** icon . The grid is added to the form and the **C1TrueDBGrid Tasks** menu appears.
3. In the **C1TrueDBGrid Tasks** menu, click **Dock in parent container** to dock the grid within the entire form. For more information on accessing the **C1TrueDBGrid Tasks** menu, see [C1TrueDBGrid Tasks Menu](#).



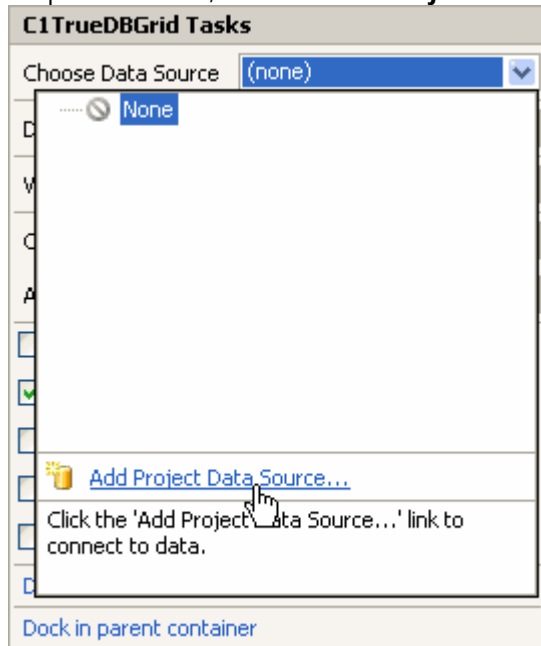
You've successfully created a simple grid application. In the next step, you'll learn how to bind the **C1TrueDBGrid** control to a data source.

Step 2 of 3: Binding True DBGrid for WinForms to a DataSet

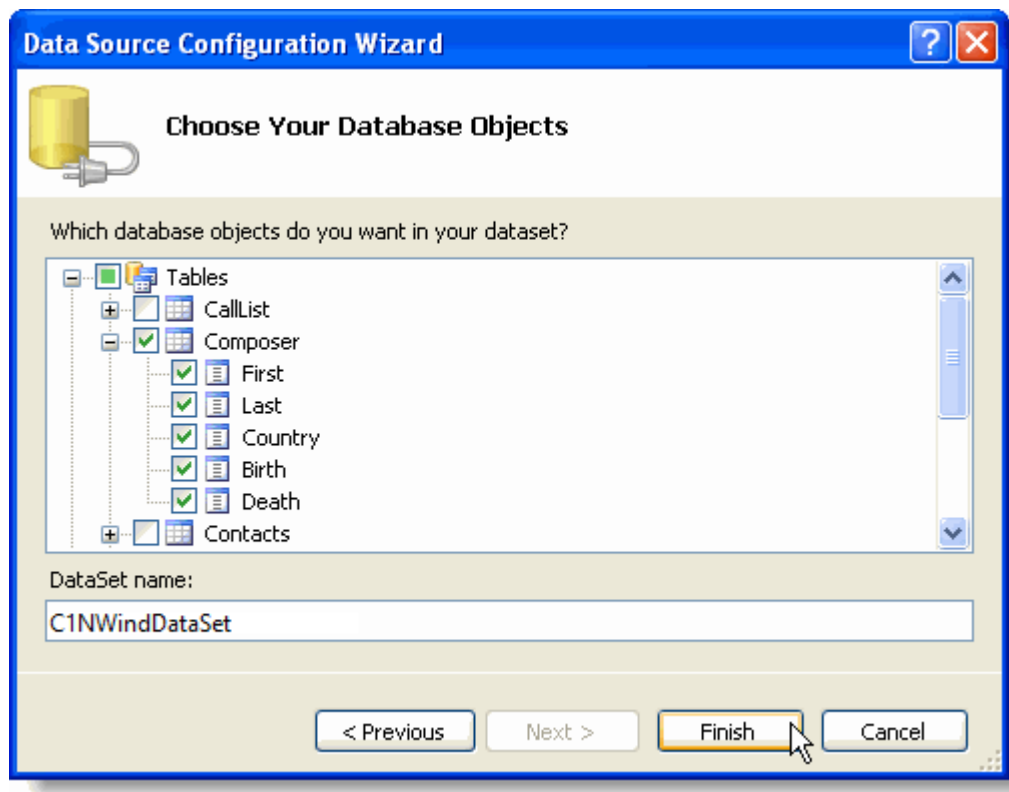
In this step, you'll learn how to bind a **C1TrueDBGrid** control to a **DataSet**. You will also learn about the basic **True DBGrid** properties and observe the run-time features of the grid. Complete the following steps to bind a

C1TrueDBGrid control to a DataSet:

1. Click **C1TrueDBGrid1**'s smart tag to open the **C1TrueDBGrid Tasks** menu, select the **Choose Data Source** drop-down arrow, and click **Add Project Data Source** to add a new data source to your project.



2. The **Data Source Configuration Wizard** appears and **Database** is selected. Click **Next**.
3. Click the **New Connection** button to locate and connect to a database.
4. Click the **Browse** button and locate **C1NWind.mdb** in the **Documents\ComponentOne Samples\Common** directory. Select it and click **Open**.
5. Click the **Test Connection** button to make sure that you have successfully connected to the database or server and click **OK**. The new string appears in the data connection drop-down list.
6. Click the **Next** button to continue. A dialog box will appear asking if you would like to add the data file to your project and modify the connection string. Click **No**.
7. In the next window, the **Yes, save the connection as** check box is checked by default and a name has been automatically entered in the text box. Click **Next** to continue.
8. In the **Choose Your Database Objects** window, you can select the tables and fields that you would like in your dataset. Select the **Composer** table.
The DataSet is given a default name in the **DataSet name** text box.



9. Click **Finish** to exit the wizard. The **DataSet**, **BindingSource** and **TableAdapter** now appear on your form.
10. Double-click the form. Notice that Visual Studio has added the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.ComposerTableAdapter.Fill(Me.DsComposer.Composer)
```

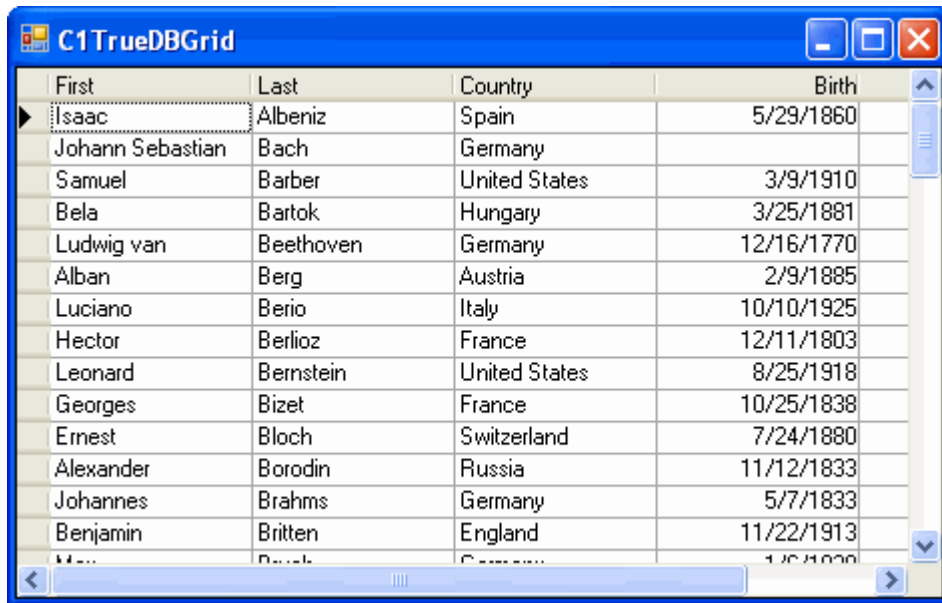
To write code in C#

C#

```
this.composerTableAdapter.Fill(this.DsComposer.Composer);
```

Run the program and observe the following:

Notice that the data from the **Composers** table is reflected in the grid:



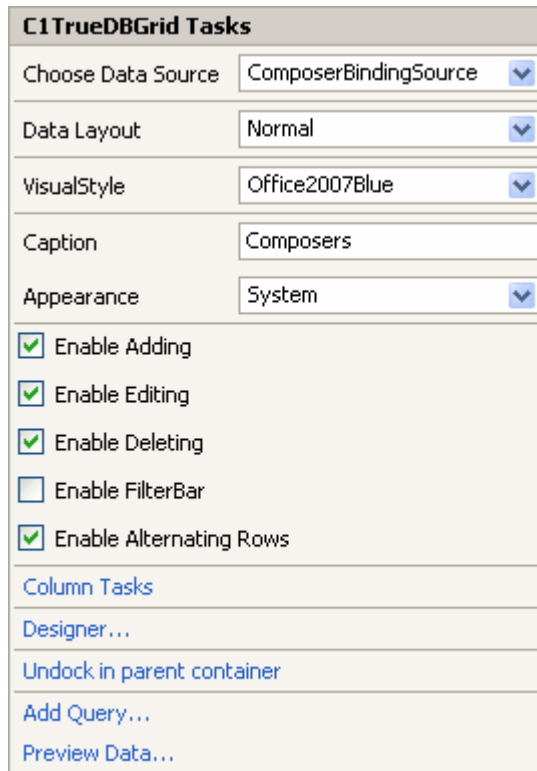
True DBGrid retrieves the database schema information from the DataSet and automatically configures itself to display all of the fields contained in the database table. Note that the field names are used as the default column headings.

Congratulations, you have successfully completed binding a **C1TrueDBGrid** control to a DataSet. In the next section you will customize the **C1TrueDBGrid** control's appearance and behavior settings.

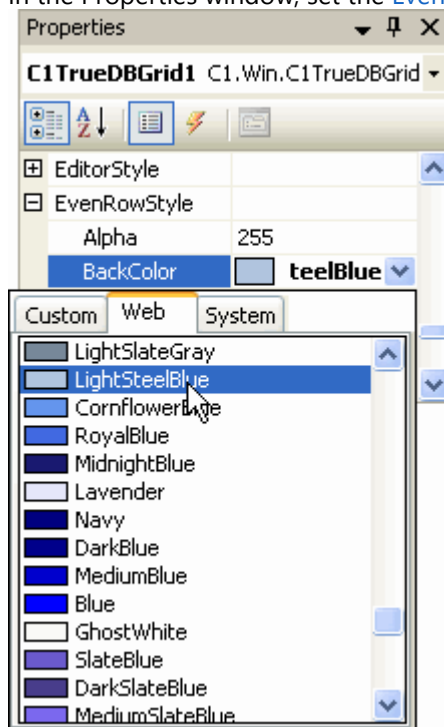
Step 3 of 3: Customizing True DBGrid for WinForms Settings

In the previous steps you've added **C1TrueDBGrid** to a project, set up the grid, and bound the grid to a data source. In this step you'll customize the grid's appearance and behavior settings. Complete the following steps:

1. Switch to **Design** view and click on **C1TrueDBGrid1**'s smart tag to open the **C1TrueDBGrid Tasks** menu.
2. In the **C1TrueDBGrid Tasks** menu set the following properties:
 - Set **Caption** property to "Composers" to add a caption to the grid.
 - Select the **Enable Adding** and **Enable Editing** check boxes to set the **AllowAddNew** and **AllowUpdate** properties to **True** and allow users to edit the grid.
 - Select **Enable Alternating Rows** to set the **AlternatingRows** property to **True**.
 - Set the **VisualStyle** property to **Office2007Blue** to set the grid's appearance.

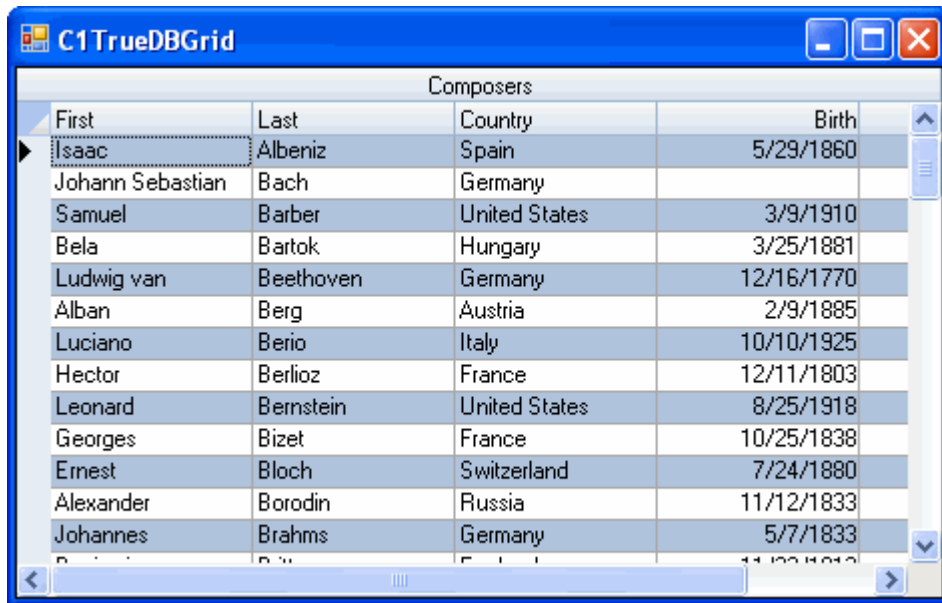


3. In the Properties window, set the [EvenRowStyle.BackColor](#) property to **LightSteelBlue**.



Run the program and observe:

You've customized the **C1TrueDBGrid** control. Notice that you've changed the appearance of the grid and can now add and edit the grid's content.



The screenshot shows a Windows application window titled "C1TrueDBGrid". Inside the window is a data grid titled "Composers". The grid has four columns: "First", "Last", "Country", and "Birth". The data is as follows:

| First | Last | Country | Birth |
|------------------|-----------|---------------|------------|
| Isaac | Albeniz | Spain | 5/29/1860 |
| Johann Sebastian | Bach | Germany | |
| Samuel | Barber | United States | 3/9/1910 |
| Bela | Bartok | Hungary | 3/25/1881 |
| Ludwig van | Beethoven | Germany | 12/16/1770 |
| Alban | Berg | Austria | 2/9/1885 |
| Luciano | Berio | Italy | 10/10/1925 |
| Hector | Berlioz | France | 12/11/1803 |
| Leonard | Bernstein | United States | 8/25/1918 |
| Georges | Bizet | France | 10/25/1838 |
| Ernest | Bloch | Switzerland | 7/24/1880 |
| Alexander | Borodin | Russia | 11/12/1833 |
| Johannes | Brahms | Germany | 5/7/1833 |

The grid has a blue header and alternating light blue and white rows. It includes a vertical scrollbar on the right and a horizontal scrollbar at the bottom.

Congratulations, you've completed the **True DBGrid** quick start! You've created a **True DBGrid for WinForms** application, bound the grid to a data source, and changed the grid's appearance and behavior settings without writing a single line of code.

True DBGrid for WinForms Top Tips

The following tips were compiled from frequently asked user questions posted in the [C1TrueDBGrid newsgroup and forum](#).

Tip 1: Use the `SetDataBinding` method to keep layout of grid intact.

If the `DataSource` is reset through code, it will show all of the data in the grid and will not keep the initial layout created with the Designer.

You can ensure that the grid layout remains as designed by using the `SetDataBinding` method with the **HoldFields** parameter set to **True**. For example:

To write code in Visual Basic

Visual Basic

```
C1TrueDBGrid1.SetDataBinding(DbDataSet, "Customer", True)
```

To write code in C#

C#

```
this.c1TrueDBGrid1.SetDataBinding(this.DbDataSet, "Customer", true);
```

Tip 2: Setting column styles through `FetchCellStyle` event.

Since columns can be moved and sorted, you should generally be careful about using the display column index and the column index as these may refer to different columns.

You can ensure that a style is associated with a particular display column. In the following example, a style is associated with a display column through the `FetchCellStyle` event:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_FetchCellStyle(ByVal sender As Object, ByVal e As  
C1.Win.C1TrueDBGrid.FetchCellStyleEventArgs) Handles C1TrueDBGrid1.FetchCellStyle  
    If Me.C1TrueDBGrid1.Splits(0).DisplayColumns(e.Col).DataColumn.Value.GetType  
Is GetType(Integer) Then  
        e.CellStyle.ForeColor = Color.Red  
    End If  
End Sub
```

To write code in C#

C#

```
private void c1TrueDBGrid1_FetchCellStyle(object sender, FetchCellStyleEventArgs e)  
{  
    if (this.c1TrueDBGrid1.Splits[0].DisplayColumns[e.Col].DataColumn.Value.GetType()  
== typeof(string))  
    {  
        e.CellStyle.ForeColor = Color.Red;  
    }  
}
```

```
}  
}
```

Tip 3: Getting current column and row number of the Grid.

It can be really useful to find out what cell a user is interacting with, or what cell is currently selected. Getting the column number and row number of the selected cell is very simple to do.

For example, the following code determines and displays the row and column number of the current cell:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Row = Me.C1TrueDBGrid1.RowContaining(C1TrueDBGrid1.Row)  
Me.C1TrueDBGrid1.Col = Me.C1TrueDBGrid1.ColContaining(C1TrueDBGrid1.Col)  
MessageBox.Show("The number of the column is " & Me.C1TrueDBGrid1.Col & " the row row  
number is " & Me.C1TrueDBGrid1.Row)
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Row = this.c1TrueDBGrid1.RowContaining(c1TrueDBGrid1.Row);  
this.c1TrueDBGrid1.Col = this.c1TrueDBGrid1.ColContaining(c1TrueDBGrid1.Col);  
MessageBox.Show("The number of the column is " + this.c1TrueDBGrid1.Col + " the row  
row number is " + this.c1TrueDBGrid1.Row);
```

Tip 4: Stop users from collapsing the grid into the normal data view when in the hierarchical data view

When the grid is in the hierarchical data view you can easily stop users from collapsing the grid back to the normal data view.

Using the [Collapse](#) event, set **e.Cancel = True** to prevent users from collapsing the expand icon. For example:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_Collapse(ByVal sender As System.Object, ByVal e As  
C1.Win.C1TrueDBGrid.BandEventArgs) Handles C1TrueDBGrid1.Collapse  
    e.Cancel = True  
End Sub
```

To write code in C#

C#

```
private void c1TrueDBGrid1_Collapse(object sender, BandEventArgs e)  
{  
    e.Cancel = true;  
}
```

Object Model

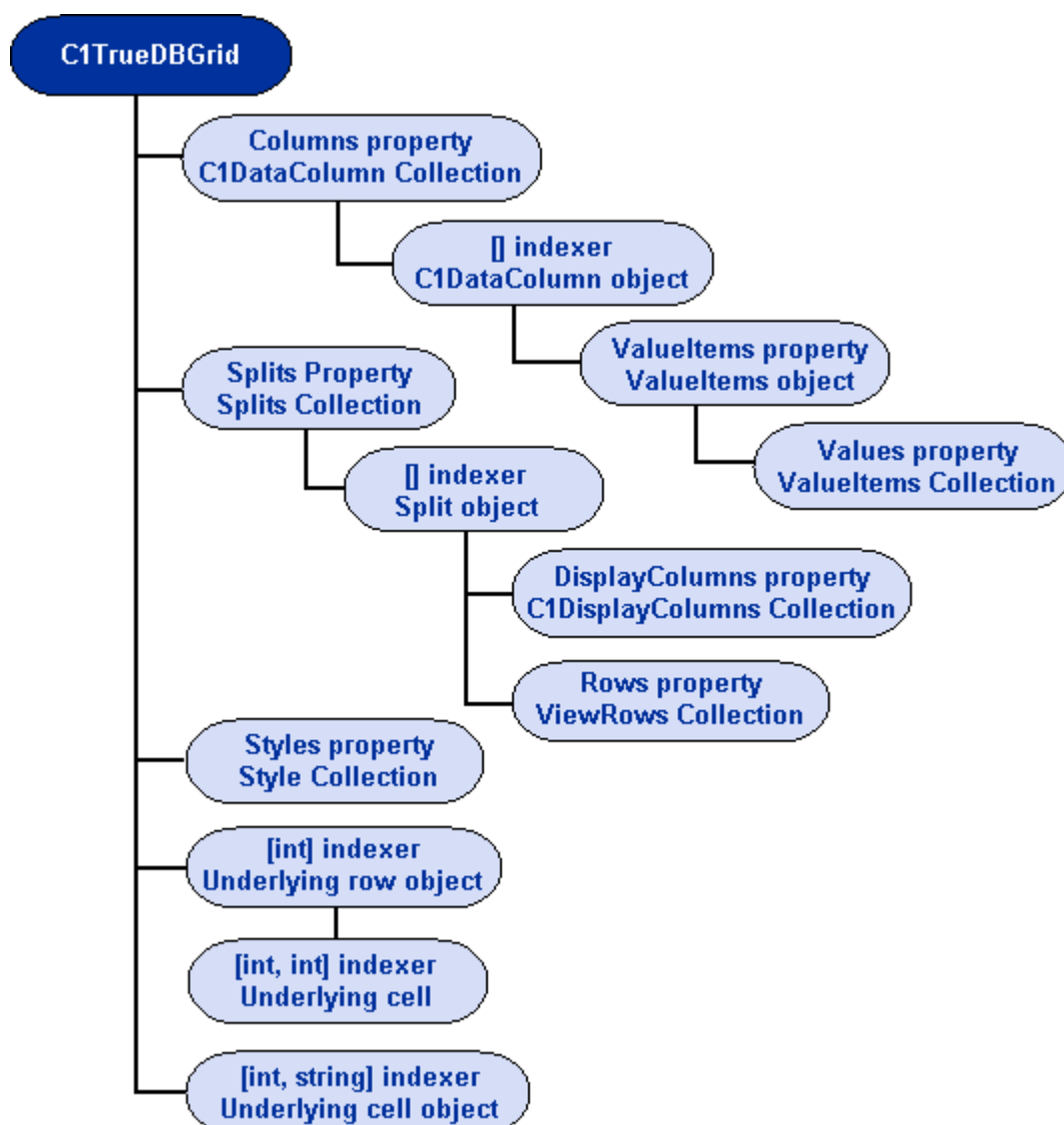
True DBGrid for WinForms was developed using the latest .NET technologies. The **True DBGrid for WinForms** controls and their programmable components are all .NET objects designed according to Microsoft specifications. If you're already familiar with the Microsoft .NET object and collection models, you'll have no problem using **True DBGrid for WinForms**.

If you're new to Visual Studio, please read [Working with Objects and Collections](#), which illustrates how to manipulate **True DBGrid for WinForms** objects in code. Although individual objects are designed to perform different tasks, the techniques used to manipulate them are the same. Once you have mastered these common programming constructs, using Visual Studio controls will be quite easy and intuitive.

Regardless of your experience level, please read the following section, as it provides a thumbnail sketch of all **True DBGrid for WinForms** objects and collections.

True DBGrid for WinForms Objects and Collections

True DBGrid for WinForms has a rich object model with the following elements:



True DBGrid for WinForms provides a rich set of properties, methods, and events that enable you to develop sophisticated database applications. The organization imposed by **True DBGrid**'s object model makes it easier to work with such a large feature set.

Objects and collections that refer to visual entities, such as columns, can be customized in the designer or in code. Objects and collections that refer to abstract entities, such as arrays and bookmarks, are only available in code.

Two controls are available in the .NET Toolbox for addition into a project:

| Control | Description |
|----------------------------------|--|
| C1TrueDBGrid | True DBGrid for WinForms grid control. |
| C1TrueDBDropDown | True DBGrid for WinForms drop-down box control. |

The namespace for **True DBGrid for WinForms** also contains definitions for the following objects:

| Object | Description |
|---|---|
| C1DataColumn | Represents a column of data within a grid. |
| C1DisplayColumn | Represents a column of data relative to a split. |
| GridLines | Represents the gridlines which separate items in the grid. |
| HBar | Represents the horizontal scroll bar and its properties. |
| PrintPreviewWinSettings | Encapsulates the print preview window and its properties. |
| PrintInfo | Encapsulates page setup and print job settings. |
| Split | Represents a group of adjacent columns that scroll as a unit. |
| Style | Encapsulates font, color, picture, and formatting information. |
| ValueItems | Encapsulates both the Values collection and ValueItem properties. |
| ValueItem | Allowable column input values, with optional translation. |
| VBar | Represents the vertical scroll bar and its properties. |

A *collection* is an object used to group similar data items, such as grid columns or styles. In general, a group of similar items in **True DBGrid for WinForms** is implemented as a collection. Since a collection is an object, it can be manipulated in code just like any other object. **True DBGrid in WinForms** exposes the following collections:

| Collection | Description |
|---|---|
| C1DataColumnCollection | Contains zero or more C1DataColumn objects in a grid. |
| C1DisplayColumnCollection | Contains zero or more C1DisplayColumn objects in a grid. |
| GroupedColumnCollection | Contains zero or more C1DataColumn objects in the grouping area. |
| SelectedRowCollection | Contains zero or more selected row indexes. |
| SelectedColumnCollection | Contains zero or more C1DataColumn objects that represent selected columns. |
| SplitCollection | Contains one or more Split objects in a grid. |
| GridStyleCollection | Contains built-in and user-defined Style objects for a grid. |
| ValueItemCollection | Contains zero or more ValueItem objects for a column. |

The following sections provide a brief overview of **True DBGrid for WinForm**'s objects and collections.

C1TrueDBGrid Class

The **C1TrueDBGrid** control is the primary object of **True DBGrid for WinForms**. Use its [C1DataColumnCollection](#) and [C1DisplayColumnCollection](#) objects to create, access, and modify the column objects that define the mappings between the grid's physical columns and the underlying database fields. Using its [SplitCollection](#) object, the grid can be divided into multiple horizontal or vertical panes to provide different views of the same data source.

C1TrueDBDropDown Class

The [C1TrueDBDropDown](#) control, which is a subset of the **C1TrueDBGrid** control, is used as a multicolumn drop-down list box for a grid column. The [C1TrueDBDropDown](#) control cannot be used as a standalone control.

In the designer, place a [C1TrueDBDropDown](#) control on a form just as you would a **C1TrueDBGrid** control. However, the drop-down control will be invisible at run time unless it is attached to a [C1DataColumn](#) object of a [C1TrueDBGrid](#) control.

To use the drop-down control, set the [DropDown](#) property of a grid column to the name of a [C1TrueDBDropDown](#) control at either in the designer or in code. At run time, when the user clicks the in-cell button for that column, the [C1TrueDBDropDown](#) control will appear below the grid's current cell. If the user selects an item from the drop-down control, the grid's current cell is updated. The [C1TrueDBDropDown](#) control also supports incremental search.

C1DataColumnCollection Class

The [C1TrueDBGrid](#) control and the [C1TrueDBDropDown](#) control both maintain a [C1DataColumnCollection](#) object to hold and manipulate [C1DataColumn](#) objects. This collection is contained under the [C1TrueDBGrid](#) object, and can be modified through the **C1TrueDBGrid Designer**. It can be accessed through the **Columns** property of the **True DBGrid for WinForms** controls.

C1DataColumn Object

Each column within a [C1TrueDBGrid](#) or [C1TrueDBDropDown](#) control is represented by two column objects, one global and one split-specific. All of the properties related to data access and formatting are contained under the [C1DataColumn](#) object. The properties of the [C1DataColumn](#) object are global in scope; changing a [C1DataColumn](#) property changes that value for all columns, even across splits. The [C1DataColumn](#) object can be accessed as follows:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns(0).Caption = "Region"
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Columns[0].Caption = "Region";
```

C1DisplayColumnCollection Class

The [C1TrueDBGrid](#) control and the [C1TrueDBDropDown](#) control both maintain a [C1DisplayColumnCollection](#) object to hold and manipulate [C1DisplayColumn](#) objects. This collection is contained under the [Split](#) object, and is available through the Split's [DisplayColumns](#) property. In addition, this collection can be modified in .NET through the **C1DisplayColumnCollection Editor**. For more information, see [Using the C1DisplayColumnCollection Editor](#).

C1DisplayColumn Class

Each split within the grid contains at least one [C1DisplayColumn](#) object. All of the properties related to a column's display are contained under this object. Unlike the [C1DataColumn](#) properties, the properties of the [C1DisplayColumn](#) object are split-specific. Changing a [C1DisplayColumn](#) property will change that value for only the specified column inside the specified split. The object can be accessed as follows:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0,0).DisplayColumns(0).Style.ForeColor =  
System.Drawing.Color.Blue
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0,0].DisplayColumns[0].Style.ForeColor =  
System.Drawing.Color.Blue;
```

GroupedColumnCollection Class

When the [DataView](#) property is set to [DataViewEnum.GroupBy](#), a grouping area is created above the grid. This collection object represents the columns ([C1DataColumn](#) object) in the grouping area. As columns are dragged into or dragged out of the grouping area, the corresponding column in the collection will be added or deleted.

SplitCollection Class

The [C1TrueDBGrid](#) control maintains a [SplitCollection](#) collection to hold and manipulate [Split](#) objects. A grid has one split by default, but may contain multiple splits. This collection is accessed using the [Splits](#) property of the [C1TrueDBGrid](#). In addition this collection can be modified in .NET through the **Split Collection Editor**. See [Using the Split Collection Editor](#) for more information.

Split Object

True DBGrid for WinForms supports Excel-like splits that divide the grid into vertical and horizontal panes to provide users with different views of the data source. Each split is represented by a [Split](#) object and contains a group of adjacent columns that scroll as a unit.

When a [C1TrueDBGrid](#) control is created, it contains one [Split](#) object by default. Many of the properties of the [Split](#) object also apply to the [C1TrueDBGrid](#) control as a whole, so there is no need to be concerned with splits until needed such as when creating fixed, nonscrolling columns. The object can be accessed as follows:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).Caption = "Split00"
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].Caption = "Split00";
```

GridStyleCollection Class

The [C1TrueDBGrid](#) and [C1TrueDBDropDown](#) controls store all built-in and user-defined [Style](#) objects in the [GridStyleCollection](#) object. Access the members of this collection by name in code, and then apply them to a grid, column, or split in order to control the appearance of the object in question. This collection is accessed using the [Styles](#) property in the **True DBGrid for WinForms** controls. In addition, this collection and its members can be modified in .NET through the [C1TrueDBGrid Style Editor](#).

Style Object

Style objects encapsulate font, color, picture, and formatting information for a [C1TrueDBGrid](#), [C1TrueDBDropDown](#), [Split](#), or [C1DisplayColumn](#) object. The [Style](#) object is a very flexible and powerful tool that provides Excel- and Word-like formatting capabilities for controlling the appearance of the grid's display.

When a [C1TrueDBGrid](#) or [C1TrueDBDropDown](#) control is created, it contains ten built-in styles. Modify the built-in styles or add custom styles either in the designer or in code. Both controls also support several optional events that use [Style](#) objects to convey formatting information on a per-cell or per-row basis. The object can be accessed as follows:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Styles("Normal").BackColor = System.Drawing.Color.Gray
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Styles["Normal"].BackColor = System.Drawing.Color.Gray;
```

ValueItems Class

The [ValueItems](#) object contains a collection and a couple of properties that can create alternate display values for database values in the grid. It can specify an allowable input value for a given [C1DataColumn](#) object, or it can also be used to translate raw data values into alternate text or graphics for display (for example, to display *Balance Due* and *Paid in Full* instead of the numeric data values 0 and 1). The [ValueItems](#) object contains display properties and a collection of [ValueItem](#) objects, the [ValueItemCollection](#). This object can be accessed as follows:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid.Columns(0).ValueItems.MaxComboItems = 5
```

To write code in C#

C#

```
this.c1TrueDBGrid.Columns[0].ValueItems.MaxComboItems = 5;
```

ValueItemCollection Class

Each [C1DataColumn](#) object within a [C1TrueDBGrid](#) or [C1TrueDBDropDown](#) control stores its set of display value/value pairs in objects called [ValueItem](#) objects. The [ValueItemCollection](#) object is a collection of these pairs. This collection can be accessed through the [Values](#) property of the [ValueItems](#) object. For instance, in order to alter the first [ValueItem](#) in the collection, the code would look like:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid.Columns(0).ValueItems.Values(0).DisplayValue = "Canada"
```

To write code in C#

C#

```
this.c1TrueDBGrid.Columns[0].ValueItems.Values[0].DisplayValue = "Canada";
```

ValueItem Class

The [ValueItem](#) object consists of two properties: [DisplayValue](#) and [Value](#). The [Value](#) property specifies the underlying value in the database and the [DisplayValue](#) property specifies the value to be displayed in the grid. These objects are contained in the [ValueItemCollection](#) object, and can be edited in .NET's **ValueItemCollection Editor**. This editor is available in the **C1TrueDBGrid Designer** under the [ValueItems](#) object. For more information, see [Using the ValueItemCollection Editor](#).

PrintInfo Class

The [PrintInfo](#) object is used to specify page layout and print job characteristics such as the name of the output device, margin settings, page headers and footers, and the number of copies to print.

The [PrintInfo](#) property of a [C1TrueDBGrid](#) control returns the object that modifies the print job.

The [PrintInfo](#) object is persistent, which means that a print layout can be defined at design time, and then recalled in code at run time.

PrintPreviewWinSettings Class

The [PrintPreviewWinSettings](#) object provides access to properties of the Print Preview window of the grid. Through this object, page headers, page footers, and other visual aspects can be set to the preview window. This object is accessed through the [PreviewInfo](#) property of the [C1TrueDBGrid](#) control.

HBar Class

The [HBar](#) object is used to specify properties of the horizontal scrollbar. Through the use of the [HScrollBar](#) property,

the developer can specify the height of the scroll bar, and whether it shows up automatically or not at all.

VBar Class

The [VBar](#) object is used to specify properties of the vertical scrollbar. Through the use of the [VScrollBar](#) property, the developer can specify the width of the scroll bar, and whether it shows up automatically or not at all.

GridLines Class

The [GridLines](#) object is used to specify characteristics of the [ColumnDivider](#) and [RowDivider](#) properties. Both the color and style of the column and row lines can be manipulated at run time or design time through the use of the [GridLines](#) object.

GridBorders Class

The [GridBorders](#) object is used to specify the characteristics of the [Borders](#) property of a [Style](#). This property sets the column borders for the cell. Through the use of this object, the developer can specify the width of each border around a cell and the color of the cell border.

SelectedRowCollection Class

When the user selects and highlights one or more rows of a [C1TrueDBGrid](#) control at run time, the row index of the selected rows are stored in the [SelectedRowCollection](#) object. In code, the [Item](#) property and [IndexOf](#) method of the collection can be used to determine which rows are selected. Also select and deselect records programmatically using its [Add](#) and [RemoveAt](#) methods.

SelectedColumnCollection Class

When the user selects and highlights one or more columns of a [C1TrueDBGrid](#) control at run time, the [C1DataColumn](#) objects for those rows are stored in the [SelectedColumnCollection](#) object. In code, use the [Item](#) property and [IndexOf](#) method of the collection to determine which rows are selected. Also select and deselect records programmatically using its [Add](#) and [RemoveAt](#) methods.

Working with Objects and Collections

This section describes how to work with objects and collections in code, with an emphasis on efficiency. Although the concepts are illustrated with **True DBGrid for WinForms** objects and collections, the same fundamentals can be applied to all Visual Studio objects and collections.

A [C1TrueDBGrid](#) object is created when a **True DBGrid for WinForms** control is placed on a form. [C1TrueDBGrid](#) objects created in Visual Studio will have default names of [C1TrueDBGrid1](#), [C1TrueDBGrid2](#), and so forth. The control name can be changed in the Properties window at design time.

Working with Collections

A [C1TrueDBGrid](#) object has eight separate collections that govern its diverse objects. Each of these collections has an associated property within the [C1TrueDBGrid](#) object that returns the collection object. This prevents the need for the developer to enter the entire collection name when using the grid in code. The following table outlines these

mappings:

| Collection | Associated Property |
|---|---|
| C1DataColumnCollection | Columns property |
| C1DisplayColumnCollection | DisplayColumns property |
| GridStyleCollection | Styles property |
| SelectedColumnCollection | SelectedCols property |
| SelectedRowCollection | SelectedRows property |
| SplitCollection | Splits property |
| ValueItemCollection | Values property |

By default, the [SplitCollection](#) object contains one [Split](#) object. The [GridStyleCollection](#) object contains ten default [Style](#) objects: **Normal**, **Heading**, **Footing**, **Selected**, **Caption**, **HighlightRow**, **EvenRow**, **OddRow**, **RecordSelector**, and **FilterBar**.

Reference an object in a collection using its zero-based index. Read or set the [Split](#) object's properties as follows:

To write code in Visual Basic

Visual Basic

```
' Read a Split object property.
variable = Me.C1TrueDBGrid1.Splits(0).Property

' Set a Split object property.
Me.C1TrueDBGrid1.Splits(0).Property = variable
```

To write code in C#

C#

```
// Read a Split object property.
variable = this.c1TrueDBGrid1.Splits[0].Property;

// Set a Split object property.
this.c1TrueDBGrid1.Splits[0].Property = variable;
```

Create a reference to an object in a collection using the collection's **Item** method. The following code creates a reference to a grid's default [Split](#) object:

To write code in Visual Basic

Visual Basic

```
' Declare Split0 as a Split object.
Dim Split0 As C1.Win.C1TrueDBGrid.Split

' Set Split0 to reference the first Split in the collection.
Split0 = Me.C1TrueDBGrid1.Splits(0)
```

To write code in C#

C#

```
// Declare Split0 as Split object.  
C1.Win.C1TrueDBGrid.Split Split0;  
  
// Set Split0 to reference the first Split in the collection.  
Split0 = this.c1TrueDBGrid1.Splits[0];
```

Note the use of the namespace qualifier in the preceding example. Using the namespace qualifier is recommended in order to resolve potential naming conflicts with other controls. For example, if another control is used in the same project that also defines an object named [Split](#), the **True DBGrid for WinForms** namespace qualifier is required, as is the namespace qualifier for the other control.

Since the **Item** method is implicit for collections, it can be omitted:

To write code in Visual Basic

Visual Basic

```
' Declare Split0 as a Split object.  
Dim Split0 As C1.Win.C1TrueDBGrid.Split  
  
' Set Split0 to reference the first Split in the collection.  
Split0 = Me.C1TrueDBGrid1.Splits(0)
```

To write code in C#

C#

```
// Declare Split0 as Split object.  
C1.Win.C1TrueDBGrid.Split Split0;  
  
// Set Split0 to reference the first Split in the collection.  
Split0 = this.c1TrueDBGrid1.Splits[0];
```

Use Split0 to read or set the [Split](#) object's properties or to execute its methods:

To write code in Visual Basic

Visual Basic

```
' Read a Split object property.  
variable = Split0.Property  
  
' Set a Split object property.  
Split0.Property = variable  
  
' Execute a Split object method.  
Split0.Method (arg1, arg2, ...)
```

To write code in C#

C#

```
// Read a Split object property.  
variable = Split0.Property;
```

```
// Set a Split object property.
Split0.Property = variable;

// Execute a Split object method.
Split0.Method (arg1, arg2, ...);
```

Very often, you need to read and set more than one of an object's properties. For example:

To write code in Visual Basic

Visual Basic

```
' Read a Split object's properties.
variable1 = Me.C1TrueDBGrid1.Splits(0,0).Property1
variable2 = Me.C1TrueDBGrid1.Splits(0,0).Property2

' Set a Split object's properties.
Me.C1TrueDBGrid1.Splits(0,0).Property1 = variable1
Me.C1TrueDBGrid1.Splits(0,0).Property2 = variable2
```

To write code in C#

C#

```
// Read a Split object's properties.
variable1 = this.c1TrueDBGrid1.Splits[0,0].Property1;
variable2 = this.c1TrueDBGrid1.Splits[0,0].Property2;

// Set a Split object's properties.
this.c1TrueDBGrid1.Splits[0,0].Property1 = variable1;
this.c1TrueDBGrid1.Splits[0,0].Property2 = variable2;
```

This code is very inefficient because the amount of times the object `C1TrueDBGrid1.Splits(0,0)` is accessed. It is more efficient to create a single reference to the object up front and use it repeatedly:

To write code in Visual Basic

Visual Basic

```
' Declare Split0 as a Split.
Dim Split0 As C1TrueDBGrid.Split

' Set Split0 to reference the first Split in the collection.
Split0 = Me.C1TrueDBGrid1.Splits.Item(0,0)

' Read a Split object's properties.
variable1 = Split0.Property1
variable2 = Split0.Property2

' Set a Split object's properties.
Split0.Property1 = variable1
Split0.Property2 = variable2
```

To write code in C#

C#

```
// Declare Split0 as Split object.
C1TrueDBGrid.Split Split0;

// Set Split0 to reference the first Split in the collection.
Split0 = this.c1TrueDBGrid1.Splits[0,0];

// Read a Split object's properties.
variable1 = Split0.Property1;
variable2 = Split0.Property2;

// Set a Split object's properties.
Split0.Property1 = variable1;
Split0.Property2 = variable2;
```

This code is much more efficient and also easier to read. If the Visual Studio application accesses collection objects frequently, the performance of your code can be improved significantly by adhering to these guidelines.

Similarly, this technique can be applied to other objects and collections of **True DBGrid**, and of Visual Studio in general. Of particular importance to the grid are the [C1DataColumn](#) and [C1DataColumnCollection](#) objects (also applies to [C1DisplayColumn](#) object):

To write code in Visual Basic

Visual Basic

```
' Declare Cols as a Columns collection object, then set it to reference
C1TrueDBGrid1's C1DataColumnCollection object.
Dim Cols As C1.Win.C1TrueDBGrid.C1DataColumnCollection
Cols = Me.C1TrueDBGrid1.Columns

' Declare Col0 as a C1DataColumn object, then set it to reference the first Column
object in the collection.
Dim Col0 As New C1.Win.C1TrueDBGrid.C1DataColumn
Col0 = Cols(0)

' Read and set the C1DataColumn object's Property1.
variable1 = Col0.Property1
Col0.Property1 = variable1

' Execute the C1DataColumn object's Method1 (declared as a Sub).
Col0.Method1 (arg1, arg2, ...)

' Execute the C1DataColumn object's Method2 (declared as a Function).
variable2 = Col0.Method2 (arg1)
```

To write code in C#

C#

```
// Declare Cols as a Columns collection object, then set it to reference
C1TrueDBGrid1's C1DataColumnCollection object.
C1.Win.C1TrueDBGrid.C1DataColumnCollection Cols;
Cols = this.c1TrueDBGrid1.Columns;
```

```
// Declare Col0 as a C1DataColumn object, then set it to reference the first Column
// object in the collection.
C1.Win.C1TrueDBGrid.C1DataColumn Col0 = new C1TrueDBGrid.DataColumn();
Col0 = Cols[0];

// Read and set the C1DataColumn object's Property1.
variable1 = Col0.Property1;
Col0.Property1 = variable1;

// Execute the C1DataColumn object's Method1 (declared as a Sub).
Col0.Method1 (arg1, arg2, ...);

// Execute the C1DataColumn object's Method2 (declared as a Function).
variable2 = Col0.Method2 (arg1);
```

Visual Basic also provides an efficient With statement for setting multiple properties of an object without explicitly assigning it to a variable. For example, the following code sets multiple properties of the first column of a grid (recall that collections are zero-based):

To write code in Visual Basic

Visual Basic

```
With Me.C1TrueDBGrid1.Columns(0)
    .Property1 = variable1
    .Property2 = variable2
End With
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Columns[0].Property1 = variable1;
this.c1TrueDBGrid1.Columns[0].Property2 = variable2;
```

Adding Members

To create and add an object to a collection, use the collection's **Add** method. The method takes an object as its only argument. For example, create more value items for a column by adding new [ValueItem](#) objects to the [ValueItemCollection](#) object:

To write code in Visual Basic

Visual Basic

```
' Create a ValueItem object.
Dim v As C1TrueDBGrid.ValueItem = new C1TrueDbGrid.ValueItem()
Me.C1TrueDBGrid1.Columns(0).ValueItems.Values.Add(v)
```

To write code in C#

C#

```
// Create a ValueItem object.  
C1TrueDBGrid.ValueItem v = new C1TrueDBGrid.ValueItem();  
this.c1TrueDBGrid1.Columns[0].ValueItems.Values.Add(v);
```

This code adds a [ValueItem](#) object to the [ValueItemCollection](#) of C1TrueDBGrid1. Alternatively, create a [ValueItem](#) object with index 1 with the Insert method:

To write code in Visual Basic

Visual Basic

```
' Create a Split object with index 1.  
Dim S As C1TrueDBGrid.ValueItem  
Me.C1TrueDBGrid1.Columns(0).ValueItems.Values.Insert(1, S)
```

To write code in C#

C#

```
// Create a Split object with index 1.  
C1TrueDBGrid.ValueItem S;  
this.c1TrueDBGrid1.Columns[0].ValueItems.Values.Insert(1, S);
```

The only object that is unable to add or remove members using the **Add** or **RemoveAt** methods is the [Split](#) object. [InsertHorizontalSplit](#) / [RemoveHorizontalSplit](#) and [InsertVerticalSplit](#) / [RemoveVerticalSplit](#) methods of the split object must be used to correctly add or remove Splits. These methods are also available in the grid's right-click context menu at design time.

Removing Members

Regardless of how a collection implements the **Add** or **Insert** methods, the syntax for removing items is the same. To remove an existing item from a collection, use the [RemoveAt](#) method:

To write code in Visual Basic

Visual Basic

```
' Remove the Split object with index 1.  
Me.C1TrueDBGrid1.Columns(0).ValueItems.Values.RemoveAt(1)
```

To write code in C#

C#

```
// Remove the Split object with index 1.  
this.c1TrueDBGrid1.Columns[0].ValueItems.Values.RemoveAt(1);
```

After this statement is executed, all splits with collection indexes greater than 1 will be shifted down by 1 to fill the place of the removed split. Note that the **RemoveAt** method's parameter is the location of the member to be removed.

Working with the Count Property

Determine the number of objects in a collection using the collection's **Count** property:

To write code in Visual Basic

Visual Basic

```
' Set a variable equal to the number of Splits in C1TrueDBGrid1.  
variable = Me.C1TrueDBGrid1.Splits.Count
```

To write code in C#

C#

```
// Set a variable equal to the number of Splits in C1TrueDBGrid1.  
variable = this.c1TrueDBGrid1.Splits.Count;
```

Iterate through all objects in a collection using the **Count** property as in the following example, which prints the **Caption** string of each **C1DataColumn** object in a grid:

To write code in Visual Basic

Visual Basic

```
For n = 0 To Me.C1TrueDBGrid1.Columns.Count - 1  
    Debug.WriteLine(Me.C1TrueDBGrid1.Columns(n).Caption)  
Next n
```

To write code in C#

C#

```
for (n = 0; n < this.c1TrueDBGrid1.Columns.Count; n++)  
{  
    Console.WriteLine(this.c1TrueDBGrid1.Columns[n].Caption);  
}
```

The **Count** property is also useful for appending and removing columns:

To write code in Visual Basic

Visual Basic

```
' Determine how many columns there are.  
Dim NumCols As Integer  
NumCols = Me.C1TrueDBGrid1.Columns.Count  
  
' Append a column to the end of the Columns collection.  
Dim C As C1TrueDBGrid.C1DataColumn = New C1TrueDBGrid.C1DataColumn()  
Me.C1TrueDBGrid1.Columns.Insert(NumCols, C)  
  
' Make the new column visible, since columns created at run time are invisible by default.  
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(C).Visible = True  
  
' The following loop removes all columns from the grid.  
While Me.C1TrueDBGrid1.Columns.Count  
    Me.C1TrueDBGrid1.Columns.RemoveAt(0)  
End While
```

To write code in C#

C#

```
// Determine how many columns there are.
int NumCols;
NumCols = this.clTrueDBGrid1.Columns.Count;

// Append a column to the end of the Columns collection.
ClTrueDBGrid.ClDataColumn C = new ClTrueDBGrid.ClDataColumn();
this.clTrueDBGrid1.Columns.Insert(NumCols, C);

// Make the new column visible, since columns created at run time are invisible by default.
this.clTrueDBGrid1.Splits[0].DisplayColumns[C].Visible = true;

// The following loop removes all columns from the grid.
while ( this.clTrueDBGrid1.Columns.Count > 0 )
{
    this.clTrueDBGrid1.Columns.RemoveAt(0);
}
```

An efficient For Each...Next statement that can be used iterate through the objects in a collection without using the **Count** property:

To write code in Visual Basic

Visual Basic

```
Dim C As ClTrueDBGrid.ClDataColumn
For Each C In Me.ClTrueDBGrid1.Columns
    Debug.WriteLine(C.Caption)
Next S
```

To write code in C#

C#

```
ClTrueDBGrid.ClDataColumn c;
foreach (c In this.clTrueDBGrid1.Columns)
{
    Console.WriteLine(c);
}
```

In fact, using the For Each...Next statement is the easiest way to iterate through the objects in a collection.

Design-Time Support

You can easily configure **True DBGrid for WinForms** at design time using the Properties window in Visual Studio. The following sections describe how to use **True DBGrid for WinForms**' design-time environment to configure the [C1TrueDBGrid](#) control. Most of the following material also applies to the [C1TrueDBDropDown](#) control since it is a subset of C1TrueDBGrid. Specific differences between the two controls are discussed at the end of this chapter.

Understanding the Object Model and Property Access

True DBGrid for WinForms supports a rich object model that reflects the organization of its visual components. Therefore, in order to customize a grid's appearance and behavior, you need to know how the Properties window and collection editors reflect the grid's object model.

A *split* is similar to the split window features of products such as Microsoft Excel and Word. Splits can be used to present data in multiple vertical or horizontal panes. These panes, or splits, can display data in different colors and fonts. The panes can scroll as a unit or individually, and they can display different sets of columns or the same set. Splits can also be used to prevent one or more columns or rows from scrolling. By default, a grid contains a single split comprising all of its columns. Note that most of the split properties are not present in the main Properties window. For example, the [AlternatingRows](#) property cannot be set without opening up the [Split Collection editor](#) and modifying the [Split](#) object, because the value of this property can vary from split to split. The term *split-specific* is used to describe such properties, since they apply to individual splits rather than the grid as a whole.

Conversely, the term *global* is used to describe properties that apply to the grid as a whole, such as [DataView](#) and [BorderStyle](#). Global properties are accessible through the Properties window, which is initially located in the lower right of the Visual Studio IDE. The latter also shows *extender* properties specific to the Visual Basic environment, such as **Align** and **Tag**.

The distinction between split-specific and global properties also extends to the two column objects which represent the columns of data within the grid. Both of these objects govern a column's properties. The [C1DataColumn](#) object contains all of the column properties related to data access and formatting. The [C1DisplayColumn](#) object contains all column properties related to the column's visual display. The [C1DataColumn](#) properties are global column properties. These are properties that apply to all of the columns in the grid, no matter their placement among the splits. For instance, when a column is added or removed, the associated [C1DataColumn](#) would be added or removed. On the other hand, the [C1DisplayColumn](#) properties are split-specific properties. Setting one of these properties in one split does not mean that the properties are then set in all splits.

Accessing Global Grid Properties

Properties which apply to the entire grid object are considered global properties. Once set these properties will remain set no matter what split-specific or column properties are set. These properties can be accessed through the Properties window. It enables easy access to all of the grid's properties and allows you to set their values at design-time. The Properties window orders the properties either categorically or alphabetically. In order to allow the user access to objects and collections, the property page supports a tree view structure where objects can be expanded to show their constituent properties.

Accessing Split-Specific Properties

In the Properties window, split properties are accessed through the [Splits](#) property. By clicking on the **ellipsis** button (...) next to the Splits node, the editor for the Split Collection will appear. This editor can be used to access all of the split-specific properties as well as the [C1DisplayColumnCollection](#) properties for the current split. For more information on using the collection editor see [Using the Split Collection Editor](#).

In addition, split-specific properties are available in the **C1TrueDBGrid Designer**. For more information see [Using the C1TrueDBGrid Designer](#).

Accessing Column Properties

In the Properties window, *global* column properties, also known as [C1DataColumn](#) properties, are accessed through the [C1DataColumnCollection](#) object property. By clicking on the **ellipsis** button (...) next to the **Columns** node in the Visual Studio Properties window, the **C1TrueDBGrid Designer** will appear. For more information on using the collection editor see [Using the C1TrueDBGrid Designer](#).

In the Visual Studio Properties window, each member of the [SplitCollection](#) exposes a [DisplayColumns](#) property, also known as the [C1DisplayColumnCollection](#) object. These [C1DisplayColumn](#) properties are split-specific properties. By clicking on the **ellipsis** button next to the **Splits** node in the Properties window, then clicking on the **ellipsis** button next to the [DisplayColumns](#) node in the editor for the Split Collection, the editor for the [C1DisplayColumnCollection](#) will be brought up. For more information on using this editor, see [Using the C1DisplayColumnCollection Editor](#).

Using the Split Collection Editor

The [SplitCollection](#) is a collection of [Split](#) objects which provides access to most of the grid's display properties and properties specific to a Split. Accessing these properties in code is done through the [C1TrueDBGrid](#) object and is demonstrated by the following:

To write code in Visual Basic

Visual Basic

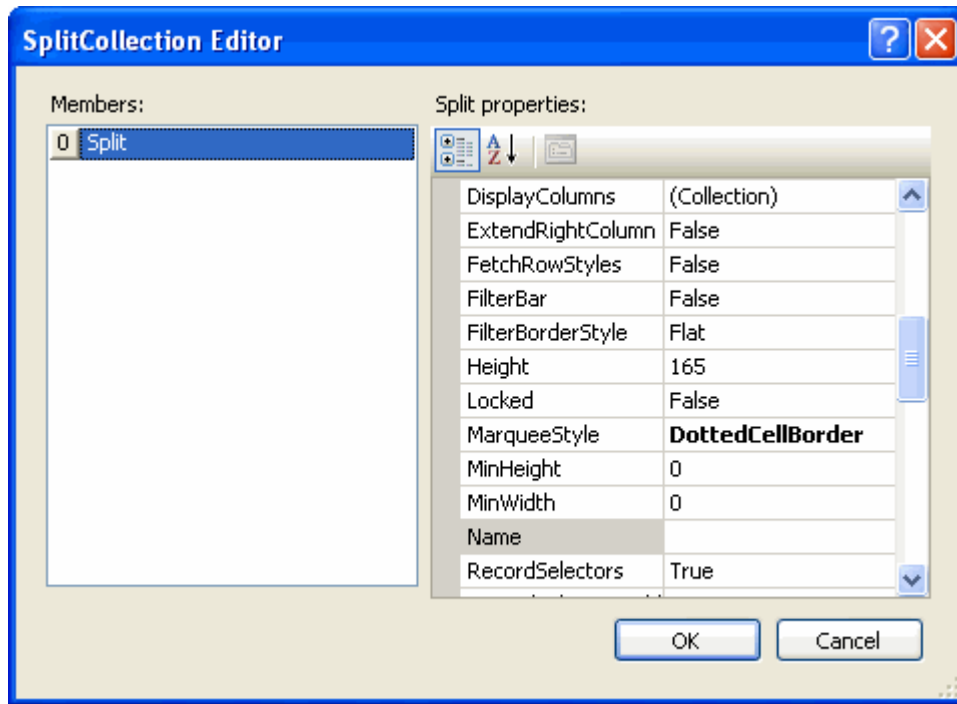
```
Me.C1TrueDBGrid1.Splits(0).AllowColMove = True
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].AllowColMove = true;
```

.NET contains useful collection editors which make the altering of a collection much easier. The [SplitCollection](#) can be modified at design-time through a .NET collection editor. The collection editor for the [SplitCollection](#) can be accessed by clicking on the **ellipsis** button (...) next to the [Splits](#) property in the Properties window. Notice that clicking on the **ellipsis** button next to the [DisplayColumns](#) property in the [SplitCollection Collection Editor](#) will bring up the [C1DisplayColumnCollection](#) editor.



Notice that the editor does not contain buttons to add and delete Splits. Even though the collection editor cannot be used to create and delete splits, this can still be accomplished at design-time. Right-clicking the grid will bring up the grid's context menu. From the context menu, choose **Design** and use the [C1TrueDBGrid Designer](#) to add or remove splits.

Splits Properties

The following [SplitCollection](#) object properties are available in the **Split Collection Editor** through the Properties window:

| Property | Description |
|---------------------------------------|---|
| AllowColMove | Gets or sets a value indicating the ability to move columns. |
| AllowColSelect | Gets or sets a value indicating the ability to select columns. |
| AllowFocus | Gets or sets a value indicating whether the split can receive focus. |
| AllowHorizontalSizing | Gets or sets a value indicating whether a user is allowed to resize horizontal splits. |
| AllowRowSelect | Gets or sets a value indicating the ability to select rows. |
| AllowRowSizing | Gets or sets how interactive row resizing is performed. |
| AllowVerticalSizing | Gets or sets a value indicating whether a user is allowed to resize vertical splits. |
| AlternatingRowStyle | Gets or sets a value indicating whether the split uses the OddRowStyle for odd-numbered rows and EvenRowStyle for even-numbered rows. |
| BorderStyle | Gets or sets the type of border rendered for a split. |
| Caption | Gets or sets the caption. |

| | |
|---------------------------------------|--|
| CaptionHeight | Gets or sets the height of the caption. |
| CaptionStyle | Gets or sets the Style object that controls the appearance of the caption area. |
| ColumnCaptionHeight | Gets or sets the height of the column captions. |
| ColumnFooterHeight | Gets or sets the height of column footers. |
| DisplayColumns | Gets a collection of C1DisplayColumn objects. |
| EditorStyle | Gets or sets the Style object that controls the appearance of the cell editor within a grid. |
| EvenRowStyle | Gets or sets the Style object that controls the appearance of an even-numbered row when using AlternatingRows . |
| ExtendRightColumn | Gets or sets a value that determines how the last column will extend to fill the dead area of the split. |
| FetchRowStyles | Gets or sets a value indicating whether the FetchRowStyle event will be raised. |
| FilterBar | Gets or sets a value indicating the visibility of the FilterBar. |
| FilterBarStyle | Gets or sets the Style object that controls the appearance of the FilterBar . |
| FilterBorderStyle | Controls the appearance of the separator for the FilterBar. |
| FooterStyle | Gets or sets the Style object that controls the appearance of column footers. |
| HeadingStyle | Gets or sets the Style object that controls the appearance of the grids column headers. |
| Height | Gets or sets the height of a split. |
| HighlightRowStyle | Gets or sets the Style object that controls the current row/cell when the MarqueeStyle is set to Highlight Row/Cell. |
| HorizontalScrollGroup | Gets or sets the group which synchronizes horizontal scrolling between splits. |
| HScrollBar | Gets the HBar object that controls the appearance of the horizontal scroll bar. |
| InactiveStyle | Gets or sets the Style object that controls the grids caption when it doesn't have focus. |
| Locked | Gets or sets a value indicating if the cells of a split can be edited. |
| MarqueeStyle | Gets or sets the MarqueeStyle for a Split. |
| MinHeight | Gets or sets the minimum height that a split can be interactively resized. |
| MinWidth | Gets or sets the minimum width that a split can be interactively resized. |
| Name | Gets or sets the name of a split. |
| OddRowStyle | Gets or sets the Style object that controls the appearance of an odd-numbered row when using AlternatingRows . |
| RecordSelectors | Gets or sets a value indicating the visibility of row headers for Split. |
| RecordSelectorStyle | Gets or sets the Style object that controls the appearance of the RecordSelectors . |
| RecordSelectorWidth | Gets or sets the width of the row headers. |
| SelectedStyle | Gets or sets the Style object that controls the appearance of selected rows |

| | |
|-------------------------------------|---|
| | and columns. |
| SplitSize | Gets or sets the size of a split. |
| SplitSizeMode | Gets or sets a value indicating how the SplitSize property is used to determine the actual size of a split. |
| SpringMode | Gets or sets a value that determines how columns will resize when the grid is resized. |
| Style | Gets or sets the root Style object for the Split. |
| VerticalScrollGroup | Gets or sets the group which synchronizes vertical scrolling between splits. |
| VScrollBar | Gets the VBar object that controls the appearance of the vertical scroll bar. |

Using the C1DisplayColumnCollection Editor

The [C1DisplayColumnCollection](#) is a collection of the column properties which relate to display, color, font, and so on. These properties are contained under the Columns identifier under the [SplitCollection](#). These properties are also split-specific; each [C1DisplayColumn](#) property can have a different value in different splits. Accessing these properties in code is done through this [SplitCollection](#), and is demonstrated by the following:

To write code in Visual Basic

Visual Basic

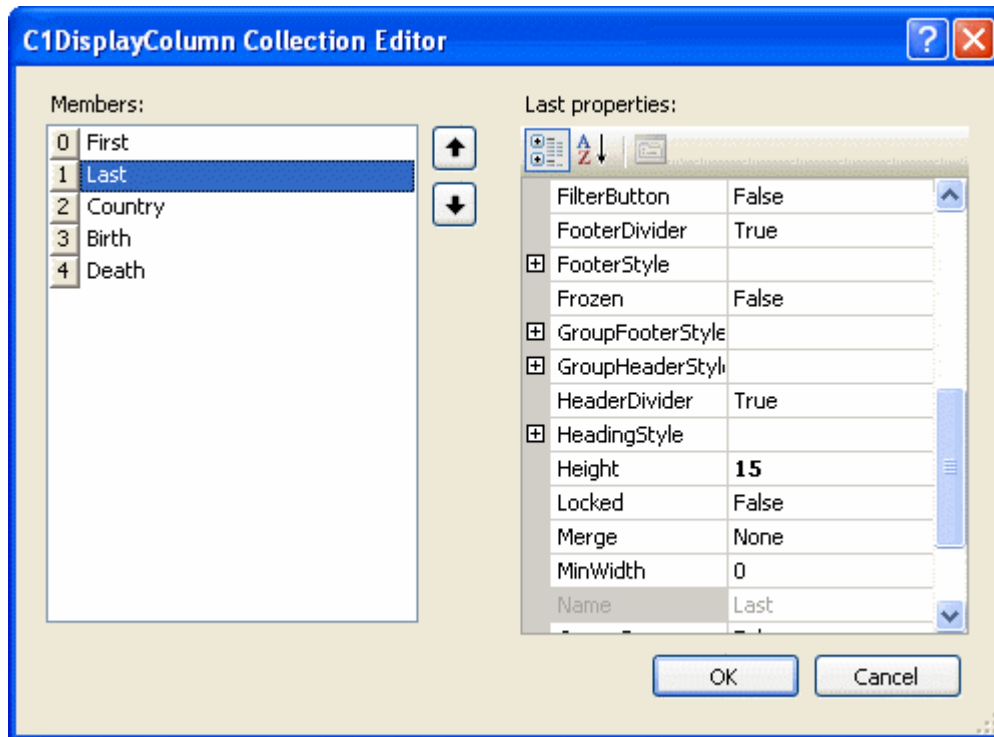
```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).Merge = True
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].Merge = true;
```

Given **True DBGrid for WinForms'** object model with its split-specific column properties and diverse collections, many of its properties might be tough to find and set efficiently. Luckily, .NET contains collection editors which help in categorizing and setting the [C1TrueDBGrid](#) control's collection properties. This editor is accessible through the **Split Collection Editor**, which can be accessed by clicking on the **ellipsis** button (...) next to the [Splits](#) property of the grid in the Properties window. In the **Split Collection Editor**, clicking on the **ellipsis** button next to the [DisplayColumns](#) property will bring up the editor.



The editor has two panes. The left pane contains the current columns in the grid under the **Members** heading. By clicking on the **Add** or **Remove** buttons the columns in the left pane can be created or deleted accordingly. The right pane contains the display-related properties for the specific column highlighted in the left pane.

Notice that there are not any add or remove buttons in the **C1DisplayColumnCollection Editor**. Due to the fact that there can be multiple **DisplayColumns** for each split in the grid, the addition or deletion of columns must occur in the **C1TrueDBGrid Designer**. This ensures that a column is added to all splits, or removed from all splits.

DisplayColumns Properties

The following [C1DisplayColumnCollection](#) object properties are available in the **C1DisplayColumnCollection Editor** in the Properties window:

| Property | Description |
|------------------------------|---|
| AllowFocus | Gets or sets a value indicating the ability of a column to receive focus. |
| AllowSizing | Gets or sets a value indicating whether column resizing is allowed. |
| AutoComplete | Gets or sets a value indicating whether the drop-down auto fills the edit portion with the matched entry. |
| AutoDropDown | Gets or sets a value indicating whether the drop-down opens automatically when a key is typed. |
| Button | Gets or sets a value indicating whether a drop-down button will be displayed in this column. |
| ButtonAlways | Gets or sets a value indicating whether buttons will be displayed when the cell does not contain focus. |
| ButtonFooter | Gets or sets a value indicating whether a column footer will act like a button. |

| Property | Description |
|----------------------------------|--|
| ButtonHeader | Gets or sets a value indicating whether a column header will act like a button. |
| ButtonText | Gets or sets a value indicating whether cells in this column look like buttons. |
| ColumnDivider | Gets or sets the style of the border drawn between columns. |
| DropDownList | Gets or sets a value indicating whether the drop-down acts like a drop-down list (text portion is not editable). |
| EditorStyle | Gets or sets the Style used for the cell editor. |
| FetchStyle | Gets or sets a value indicating whether the FetchCellStyle event will be raised for a column. |
| FilterButton | Gets or sets a value indicating whether a drop-down button will be displayed in this column. |
| FooterDivider | Gets or sets a value indicating whether to display the column divider in the footer area. |
| FooterStyle | Gets or sets the Style object that controls the appearance of column footers. |
| Frozen | Gets or sets a value indicating whether the column scrolls. |
| GroupFooterStyle | Gets or sets the Style used to render the cell in the grouped footer row. |
| GroupHeaderStyle | Gets or sets the Style used to render the cell in the grouped header row. |
| HeaderDivider | Gets or sets a value indicating whether to display the column divider in the header area. |
| HeadingStyle | Gets or sets the Style that controls the appearance of the column headers. |
| Height | Gets or sets the height of the column. |
| Locked | Gets or sets a value indicating whether editing is permitted in a column. |
| Merge | Gets or sets a value indicating whether contiguous like-value cells of this column are merged into one large cell. |
| MinWidth | Gets or sets the minimum width a column can be resized to when in SpringMode . |
| Name | Gets the caption of the associated C1DataColumn objects. |
| OwnerDraw | Gets or sets a value indicating whether cells in this column are drawn by the user in the OwnerDrawCell event. |
| Style | Gets or sets the root Style for this column. |
| Visible | Gets or sets a value indicating the visibility of a column. |
| Width | Gets or sets the width of a column. |

Using the ValueItemCollection Editor

The [ValueItemCollection](#) is a collection of values and display values which allows for translated data within a column.

This collection object can be accessed through **C1DataColumn.ValueItems.Values** property. Accessing these properties in code is done through this collection, and is demonstrated by the following:

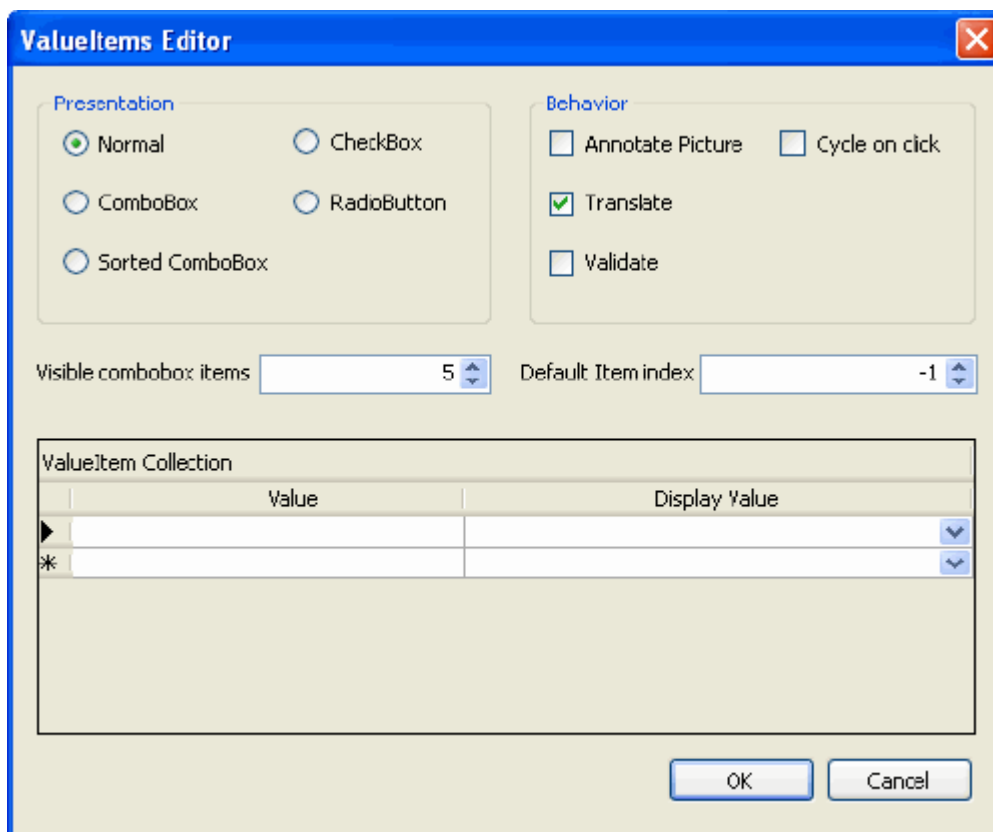
To write code in Visual Basic

```
Visual Basic
Me.C1TrueDBGrid1.Columns(0).ValueItems.Values
```

To write code in C#

```
C#
this.c1TrueDBGrid1.Columns[0].ValueItems.Values;
```

In order to make these properties more easily modifiable, there is a **ValueItem Collection Editor** which enables the user to add **ValueItems**, remove **ValueItems**, and alter their **Value** and **DisplayValue** properties. This editor is accessible through the Properties window. Clicking the **ellipsis** button (...) next to the **Columns** item in the Properties window will bring up the **C1TrueDBGrid Designer**; then expanding the **ValueItems** node will expose the **ValueItems** collection items. Clicking on the **ellipsis** button next to the **ValueItems** node will bring up the **ValueItems Editor**:



Using the C1TrueDBGrid Style Editor

The Style collection is a collection of Microsoft Word-like styles which can associate certain sections for the grid with a style. The Styles collection is located under the **C1TrueDBGrid** object, and contains individual Style objects as its members. Accessing the individual Style objects and their properties in code is done through this collection, and is demonstrated by the following:

To write code in Visual Basic

Visual Basic

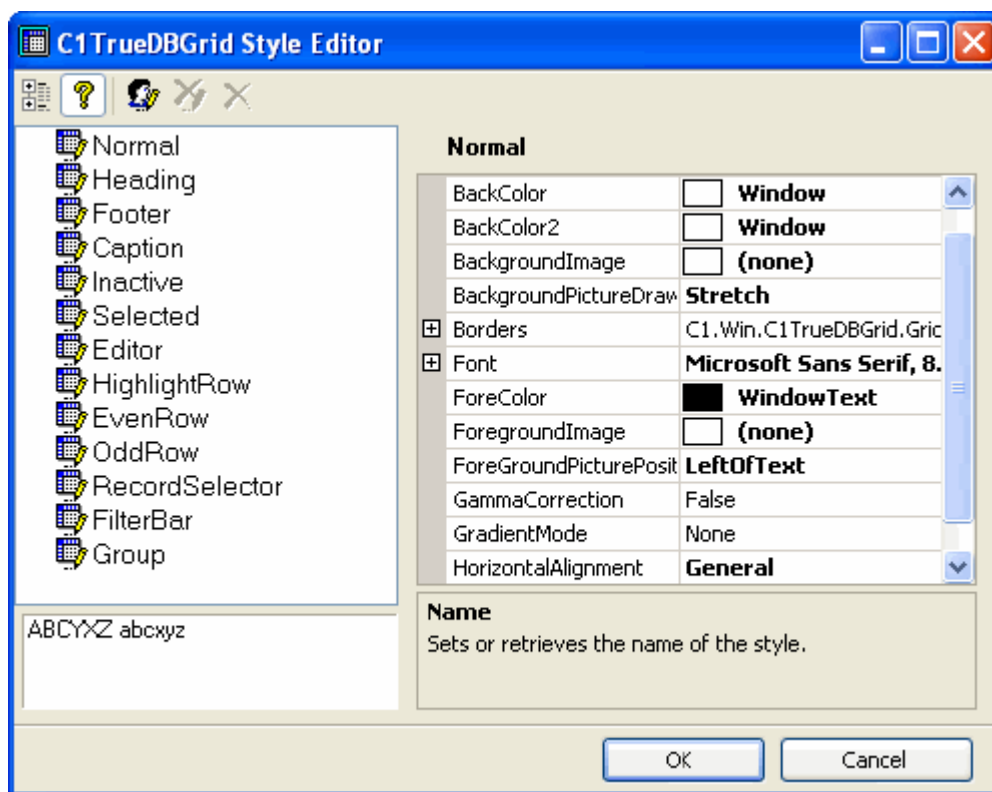
```
Me.C1TrueDBGrid1.Styles("Normal").WrapText = False
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Styles["Normal"].WrapText = false;
```

In order to make these properties more easily modifiable, there is a **C1TrueDBGrid Style Editor** which enables the user to add styles and modify the properties of existing styles. The **C1TrueDBGrid Style Editor** is available in the Properties window. Clicking on the **ellipsis** button (...) next to the **Styles** node in the Properties window will bring up the editor.



Using the C1TrueDBGrid Designer

The normal method of modifying the properties of **DisplayColumns**, **DataColumns**, and **Splits** through the property editor may seem like a complex and probably more than a bit confusing process. Keeping track of **DataColumns** and **DisplayColumns** is a task in and of itself. But to make this whole process much easier the **C1TrueDBGrid** control contains a Designer which has been constructed for ease of use.

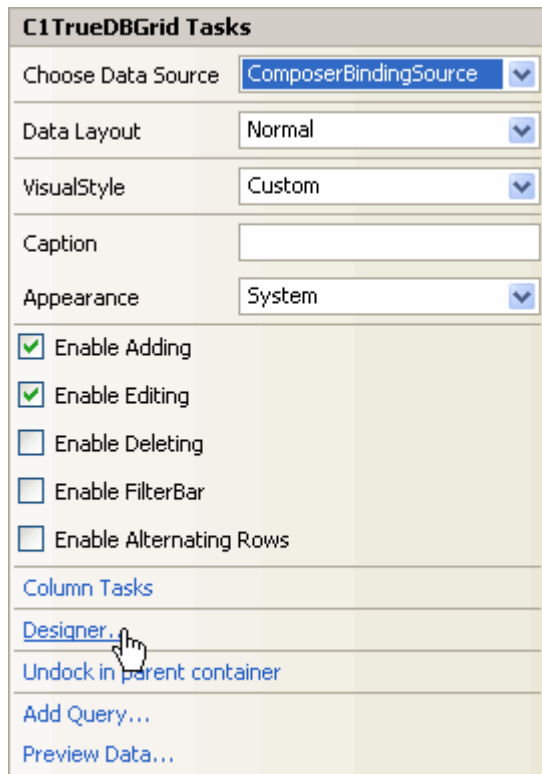
Accessing the C1TrueDBGrid Designer

The **C1TrueDBGrid Designer** can be accessed either through the **C1TrueDBGrid Tasks** menu, the **Columns** property,

or the context menu.

Through the C1TrueDBGrid Tasks Menu

To access the **C1TrueDBGrid Designer** through the **C1TrueDBGrid Tasks** menu, click the smart tag in the upper right corner of **C1TrueDBGrid** to open the **C1TrueDBGrid Tasks** menu, and select **Designer**.



Through the Columns Property

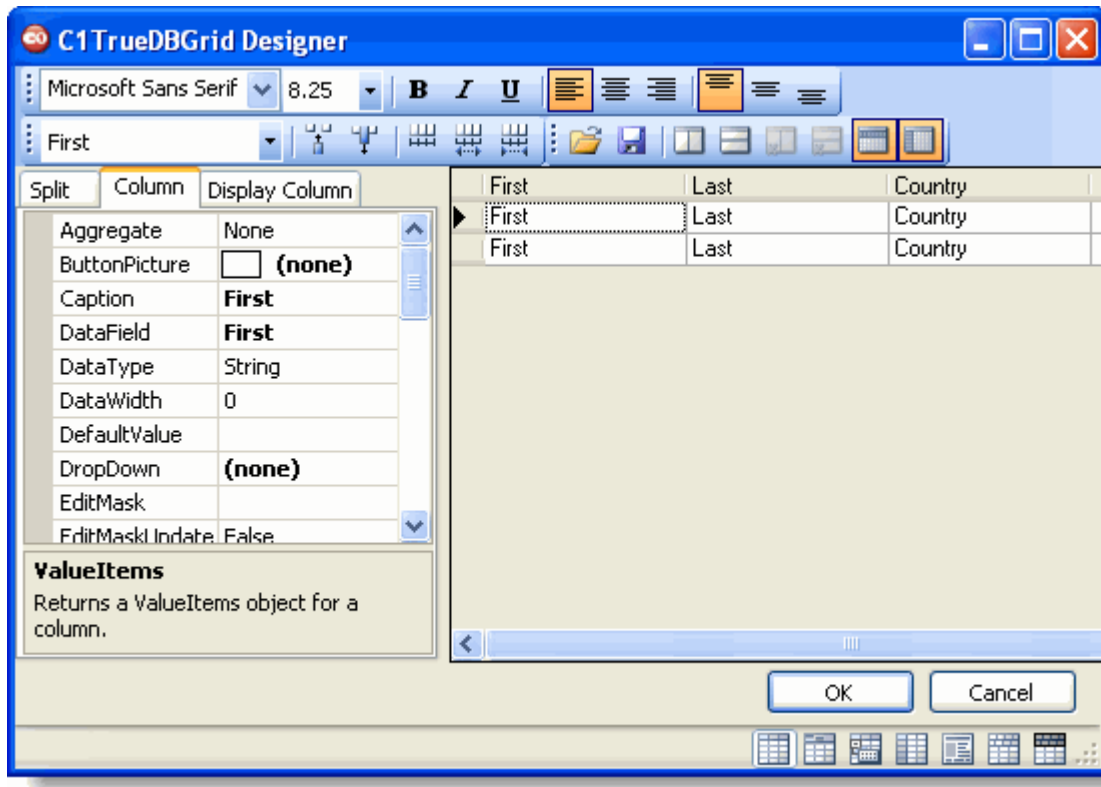
To access the **C1TrueDBGrid Designer** through the **Columns** property, click the **ellipsis** button next to the **Columns** property in the Properties window.

Through the Context Menu

To access the **C1TrueDBGrid Designer** through the context menu, right-click the **C1TrueDBGrid** control on the form and select **Design** from the context menu.

C1TrueDBGrid Designer Elements

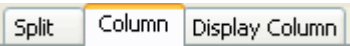
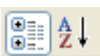
This designer allows grid columns to be set up easily at design time instead of having to write code. Just select the grid, then right-click to bring up the context menu, and then click the **Design** menu item. This will bring up the **C1TrueDBGrid Designer** shown below:










The editor displays the grid columns in a window on the right and the properties for these columns on the left. The tabs above the Properties window define which set of properties, **DataColumn**, **DisplayColumn**, or **Split**, are displayed in the properties grid.

The editor performs the following actions:

- **Reorder Columns:** Move columns to new positions by dragging them by the header cells with the mouse.
- **Adjust Column Widths:** Adjust column widths by dragging the right edge of the header cells with the mouse. You can also select multiple columns by SHIFT-clicking the header cells, and then set all column widths at once using the property grid. Setting the column width to -1 restores the default width.
- **Set Column Properties:** Whenever one or more columns are selected, their properties can be viewed and edited in the property grid on the left of the editor.
- **Insert or Remove Columns:** Use the toolbar to insert columns before or after the selection (useful mostly in unbound mode), or to remove columns.
- **Use the Toolbar to Perform Common Tasks:** The table below describes the function of the buttons on the toolbar:

| Element | Description |
|---|---|
|  | These tabs above the property grid determine which set of properties are available for modification in the designer. The tabs allow you to choose between the DataColumns property set that contains data-related column properties, the DisplayColumns property set that contains display-related column properties, and the Split property set that contains split-related properties. |
|  | These toggle buttons control the display of the property grid. The left button one indicates that the properties for the selected columns are displayed in categorized order. The right button indicates whether the properties for the selected columns are displayed in alphabetical order. |
| | These buttons set the column widths for the grid. The left button sets all |

| Element | Description |
|---|--|
|  | the columns to have the same width, the center button increases the width of the selected column (the column with focus in the grid), and the right button decreases the width of the selected column (the column with focus in the grid). |
|  | These buttons add, insert, and delete columns from the grid. The first adds columns to the grid, the second button inserts columns in the grid, and the third button deletes columns from the grid. |
|  | The drop-down box sets which column receives focus. By choosing a column from the drop-down list, the associated properties for this column will appear in the property grid to the left. |
|  | These buttons set the vertical alignment of the selected column. The first button aligns all column content to the top. The second button aligns all column content to the center, and the third button aligns all column content to the bottom. |
|  | Align column content to the left, center, or right. These buttons only affect the scrollable area of the grid. To set the alignment for the header columns, select the columns and set the TextAlignFixed property. |
|  | These buttons add or remove vertical or horizontal splits. The first button adds a vertical split to the grid, while the second one adds a horizontal split. The third button removes a vertical split, while the fourth one removes a horizontal split. |
|  | These buttons set the DataView property of the table. The buttons set the DataView property to Normal , GroupBy , Hierarchial , Inverted , Form , MultipleLines , and MultipleLinesFixed , respectively. See Data Display for more information. |

Splits Properties

The following [SplitCollection](#) object properties are available in the **C1TrueDBGrid Designer** through the **Split** tab:

| Property | Description |
|---------------------------------------|--|
| AllowColMove | Gets or sets a value indicating the ability to move columns. |
| AllowColSelect | Gets or sets a value indicating the ability to select columns. |
| AllowFocus | Gets or sets a value indicating whether the split can receive focus. |
| AllowHorizontalSizing | Gets or sets a value indicating whether a user is allowed to resize horizontal splits. |
| AllowRowSelect | Gets or sets a value indicating the ability to select rows. |
| AllowRowSizing | Gets or sets how interactive row resizing is performed. |
| AllowVerticalSizing | Gets or sets a value indicating whether a user is allowed to resize vertical splits. |

| | |
|---------------------------------------|---|
| AlternatingRowStyle | Gets or sets a value indicating whether the split uses the OddRowStyle for odd-numbered rows and EvenRowStyle for even-numbered rows. |
| BorderStyle | Gets or sets the type of border rendered for a split. |
| Caption | Gets or sets the caption. |
| CaptionHeight | Gets or sets the height of the caption. |
| CaptionStyle | Gets or sets the Style object that controls the appearance of the caption area. |
| ColumnCaptionHeight | Gets or sets the height of the column captions. |
| ColumnFooterHeight | Gets or sets the height of column footers. |
| DisplayColumns | Gets a collection of C1DisplayColumn objects. |
| EditorStyle | Gets or sets the Style object that controls the appearance of the cell editor within a grid. |
| EvenRowStyle | Gets or sets the Style object that controls the appearance of an even-numbered row when using AlternatingRows . |
| ExtendRightColumn | Gets or sets a value that determines how the last column will extend to fill the dead area of the split. |
| FetchRowStyles | Gets or sets a value indicating whether the FetchRowStyle event will be raised. |
| FilterBar | Gets or sets a value indicating the visibility of the FilterBar. |
| FilterBarStyle | Gets or sets the Style object that controls the appearance of the FilterBar . |
| FilterBorderStyle | Controls the appearance of the separator for the FilterBar. |
| FooterStyle | Gets or sets the Style object that controls the appearance of column footers. |
| HeadingStyle | Gets or sets the Style object that controls the appearance of the grids column headers. |
| Height | Gets or sets the height of a split. |
| HighlightRowStyle | Gets or sets the Style object that controls the current row/cell when the MarqueeStyle is set to Highlight Row/Cell. |
| HorizontalScrollGroup | Gets or sets the group which synchronizes horizontal scrolling between splits. |
| HScrollBar | Gets the HBar object that controls the appearance of the horizontal scroll bar. |
| InactiveStyle | Gets or sets the Style object that controls the grids caption when it doesn't have focus. |
| Locked | Gets or sets a value indicating if the cells of a split can be edited. |
| MarqueeStyle | Gets or sets the MarqueeStyle for a Split. |
| MinHeight | Gets or sets the minimum height that a split can be interactively resized. |
| MinWidth | Gets or sets the minimum width that a split can be interactively resized. |

| | |
|---------------------|--|
| Name | Gets or sets the name of a split. |
| OddRowStyle | Gets or sets the Style object that controls the appearance of an odd-numbered row when using AlternatingRows . |
| RecordSelectors | Gets or sets a value indicating the visibility of row headers for Split. |
| RecordSelectorStyle | Gets or sets the Style object that controls the appearance of the RecordSelectors . |
| RecordSelectorWidth | Gets or sets the width of the row headers. |
| SelectedStyle | Gets or sets the Style object that controls the appearance of selected rows and columns. |
| SplitSize | Gets or sets the size of a split. |
| SplitSizeMode | Gets or sets a value indicating how the SplitSize property is used to determine the actual size of a split. |
| SpringMode | Gets or sets a value that determines how columns will resize when the grid is resized. |
| Style | Gets or sets the root Style object for the Split. |
| VerticalScrollGroup | Gets or sets the group which synchronizes vertical scrolling between splits. |
| VScrollBar | Gets the VBar object that controls the appearance of the vertical scroll bar. |

C1DataColumn Properties

The following [C1DataColumnCollection](#) object properties are available in the **C1TrueDBGrid Designer** through the **Column** tab:

| Property | Description |
|--------------------------------------|---|
| Aggregate | Gets or sets the type of aggregate computed for a grouped row. |
| ButtonPicture | Gets or sets the image shown in a drop-down button in a column. |
| Caption | Gets or sets the text in the column header. |
| DataField | Gets or sets the database field name for a column. |
| DataWidth | Gets or sets the maximum number of characters which may be entered for cells in this column. |
| DefaultValue | Gets or sets the default value for a column when a new row is added by the grid. |
| DropDown | Gets or sets the C1TrueDBDropDown control associated with this column. |
| EditMask | Gets or sets the edit mask for a column. |
| EditMaskUpdate | Gets or sets a value indicating whether literal characters in the edit mask are stored to the underlying data source. |
| EnableDateTimeEditor | Gets or sets the characters that should be escaped when applying the filter criteria to the data source. |
| FilterButtonPicture | Gets or sets the image show in the filter button for the column. |

| Property | Description |
|--------------------------------|--|
| FilterDropdown | Gets or sets a value indicating whether a drop-down list is displayed in the filter cell that lists all the values of the field. |
| FilterEscape | Gets or sets the characters that should be escaped when applying the filter criteria to the data source. |
| FilterKeys | Gets or sets the key used to initiate the filtering operation as the user types in the FilterBar . |
| FilterOperator | Gets or sets the operator that is used for a filter expression. |
| FilterText | Gets or sets the data associated with the value of the filter for a column. |
| FooterText | Gets or sets the text displayed in the column footer. |
| GroupInfo | Gets or sets the GroupInfo associated with this column. |
| Level | Gets or sets the level of this column in a hierarchical data source. |
| NumberFormat | Gets or sets the formatting string for a column. |
| SortDirection | Gets or sets the state of the sorting glyph in the column caption. |
| ValueItems | Gets the ValueItems object for this column. |

DisplayColumns Properties

The following [C1DisplayColumnCollection](#) object properties are available in the **C1TrueDBGrid Designer** through the **Display Column** tab:

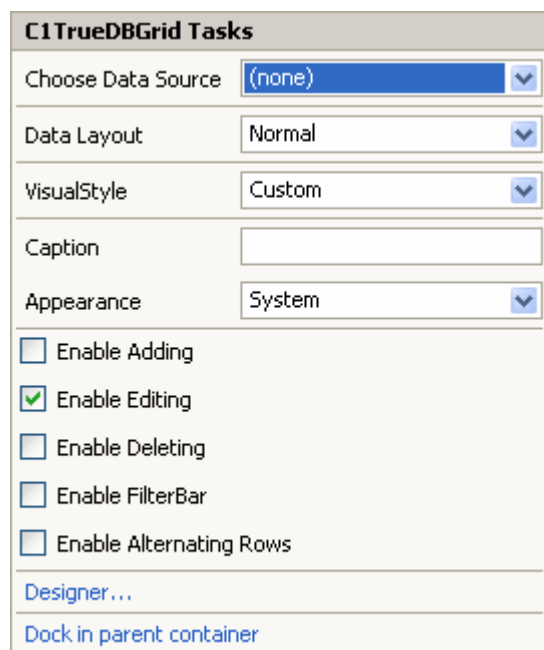
| Property | Description |
|-------------------------------|---|
| AllowFocus | Gets or sets a value indicating the ability of a column to receive focus. |
| AllowSizing | Gets or sets a value indicating whether column resizing is allowed. |
| AutoComplete | Gets or sets a value indicating whether the drop-down auto fills the edit portion with the matched entry. |
| AutoDropDown | Gets or sets a value indicating whether the drop-down opens automatically when a key is typed. |
| Button | Gets or sets a value indicating whether a drop-down button will be displayed in this column. |
| ButtonAlways | Gets or sets a value indicating whether buttons will be displayed when the cell does not contain focus. |
| ButtonFooter | Gets or sets a value indicating whether a column footer will act like a button. |
| ButtonHeader | Gets or sets a value indicating whether a column header will act like a button. |
| ButtonText | Gets or sets a value indicating whether cells in this column look like buttons. |
| ColumnDivider | Gets or sets the style of the border drawn between columns. |

| Property | Description |
|----------------------------------|--|
| DropDownList | Gets or sets a value indicating whether the drop-down acts like a drop-down list (text portion is not editable). |
| EditorStyle | Gets or sets the Style used for the cell editor. |
| FetchStyle | Gets or sets a value indicating whether the FetchCellStyle event will be raised for a column. |
| FilterButton | Gets or sets a value indicating whether a drop-down button will be displayed in this column. |
| FooterDivider | Gets or sets a value indicating whether to display the column divider in the footer area. |
| FooterStyle | Gets or sets the Style object that controls the appearance of column footers. |
| Frozen | Gets or sets a value indicating whether the column scrolls. |
| GroupFooterStyle | Gets or sets the Style used to render the cell in the grouped footer row. |
| GroupHeaderStyle | Gets or sets the Style used to render the cell in the grouped header row. |
| HeaderDivider | Gets or sets a value indicating whether to display the column divider in the header area. |
| HeadingStyle | Gets or sets the Style that controls the appearance of the column headers. |
| Height | Gets or sets the height of the column. |
| Locked | Gets or sets a value indicating whether editing is permitted in a column. |
| Merge | Gets or sets a value indicating whether contiguous like-value cells of this column are merged into one large cell. |
| MinWidth | Gets or sets the minimum width a column can be resized to when in SpringMode . |
| Name | Gets the caption of the associated C1DataColumn objects. |
| OwnerDraw | Gets or sets a value indicating whether cells in this column are drawn by the user in the OwnerDrawCell event. |
| Style | Gets or sets the root Style for this column. |
| Visible | Gets or sets a value indicating the visibility of a column. |
| Width | Gets or sets the width of a column. |

C1TrueDBGrid Tasks Menu

In the **C1TrueDBGrid Tasks** menu, you can quickly and easily choose a data source, change data layout, set a visual style, add a grid caption, customize the appearance of the grid, dock the grid on the form, and access the **C1TrueDBGrid Designer**, as well as set the following properties: [AllowAddNew](#), [AllowUpdate](#), [AllowDelete](#), [FilterBar](#), and [AlternatingRows](#).

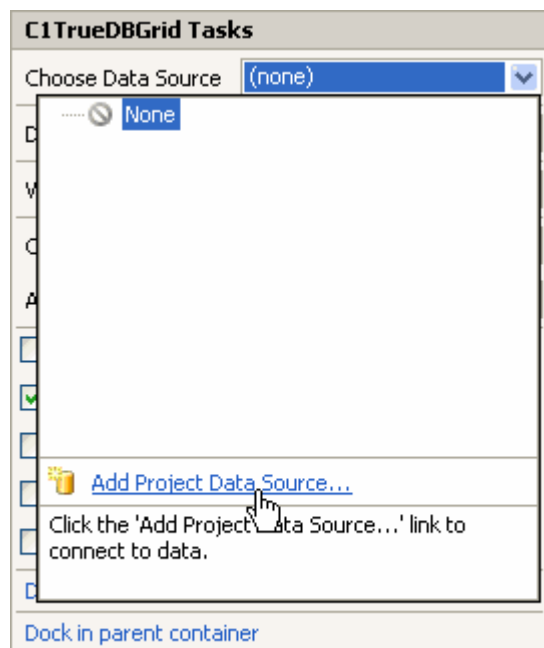
To access the **C1TrueDBGrid Tasks** menu, click the smart tag (🔗) in the upper right corner of the grid. This will open the **C1TrueDBGrid Tasks** menu:



The **C1TrueDBGrid Tasks** menu operates as follows:

Choose Data Source

Clicking the drop-down arrow in the **Choose Data Source** box opens a list of available data sources and allows you to add a new data source. To add a new data source to the project, click **Add Project Data Source** to open the **Data Source Configuration Wizard**.



After a data source is selected, three more options are added to the **C1TrueDBGrid Tasks** menu: **Column Tasks**, **Add Query**, and **Preview Data**.

C1TrueDBGrid Tasks

Choose Data Source: **ComposerBindingSource**

Data Layout: **Normal**

VisualStyle: **Custom**

Caption:

Appearance: **System**

☐ Enable Adding

☒ Enable Editing

☐ Enable Deleting

☐ Enable FilterBar

☐ Enable Alternating Rows

[Column Tasks](#)

[Designer...](#)

[Dock in parent container](#)

[Add Query...](#)

[Preview Data...](#)

Data Layout

Clicking the drop-down arrow in the **Data Layout** box opens a list of different [DataView](#) property options, such as **Normal**, **Inverted**, **Form**, **GroupBy**, **MultipleLines**, **Hierarchical**, and **MultipleLinesFixed**. For more information on the different data views, see [Data Display](#).

C1TrueDBGrid Tasks

Choose Data Source: **(none)**

Data Layout: **Normal**

VisualStyle:

Caption:

Appearance:

☐ Enable Adding

☒ Enable Editing

☐ Enable Deleting

☐ Enable FilterBar

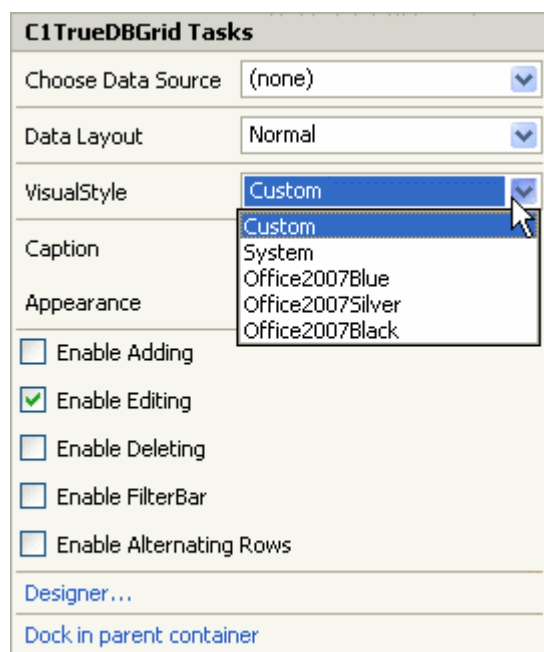
☐ Enable Alternating Rows

[Designer...](#)

[Dock in parent container](#)

VisualStyle

Clicking the drop-down arrow in the **VisualStyle** box opens a list of different [VisualStyle](#) property options, such as **Custom**, **System**, **Office2007Blue**, **Office2007Silver**, and **Office2007Black**. For more information on the different visual styles, see [Visual Styles](#).

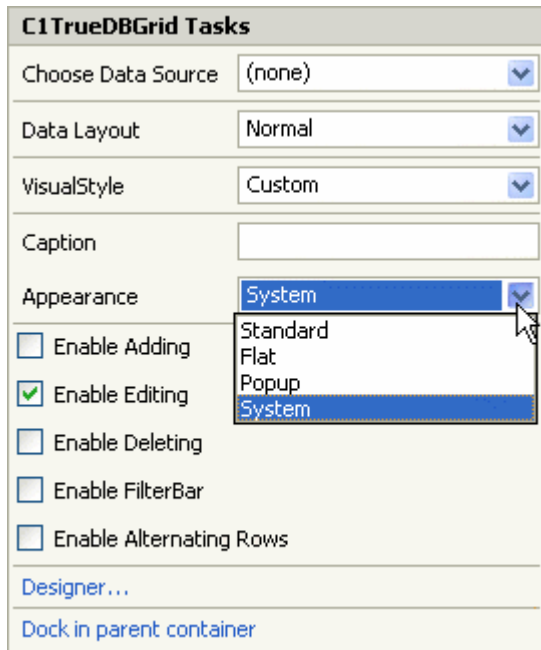


Caption

Entering a caption into the **Caption** box sets the [Caption](#) property for the grid.

Appearance

Clicking the drop-down arrow in the **Appearance** box opens a list of different [FlatStyle](#) property options, such as **Standard**, **Flat**, **Popup**, and **System**. For more information on the different control appearance options, see [Three-Dimensional vs. Flat Display](#).



Enable Adding

Selecting the **Enable Adding** check box sets the [AllowAddNew](#) property to **True**, and allows adding new rows to the grid. The default is unchecked.

Enable Editing

Selecting the **Enable Editing** check box sets the [AllowUpdate](#) property to **True**, and allows editing of the grid. The default is checked.

Enable Deleting

Selecting the **Enable Deleting** check box sets the [AllowDelete](#) property to **True**, and allows deleting rows in the grid. The default is unchecked.

Enable FilterBar

Selecting the **Enable FilterBar** check box sets the [FilterBar](#) property to **True**, and displays the [FilterBar](#) at the top of the grid. The default is unchecked.

Enable Alternating Rows

Selecting the **Enable Alternating Rows** check box sets the ["C1.Win.C1TrueDBGrid.4~C1.Win.C1TrueDBGrid.C1TrueDBGrid~AlternatingRows.html">AlternatingRows](#) property to **True**, and displays alternating row colors. The default it unchecked.

Column Tasks (available only when bound to a data source)

Clicking **Column Tasks** opens the **Column Tasks** menu. For details on the **Column Tasks** menu, see [Column Tasks Menu](#).

Designer

Clicking **Designer** opens the **C1TrueDBGrid Designer**. For more information on using the **C1TrueDBGrid Designer**, see [Using the C1TrueDBGrid Designer](#).

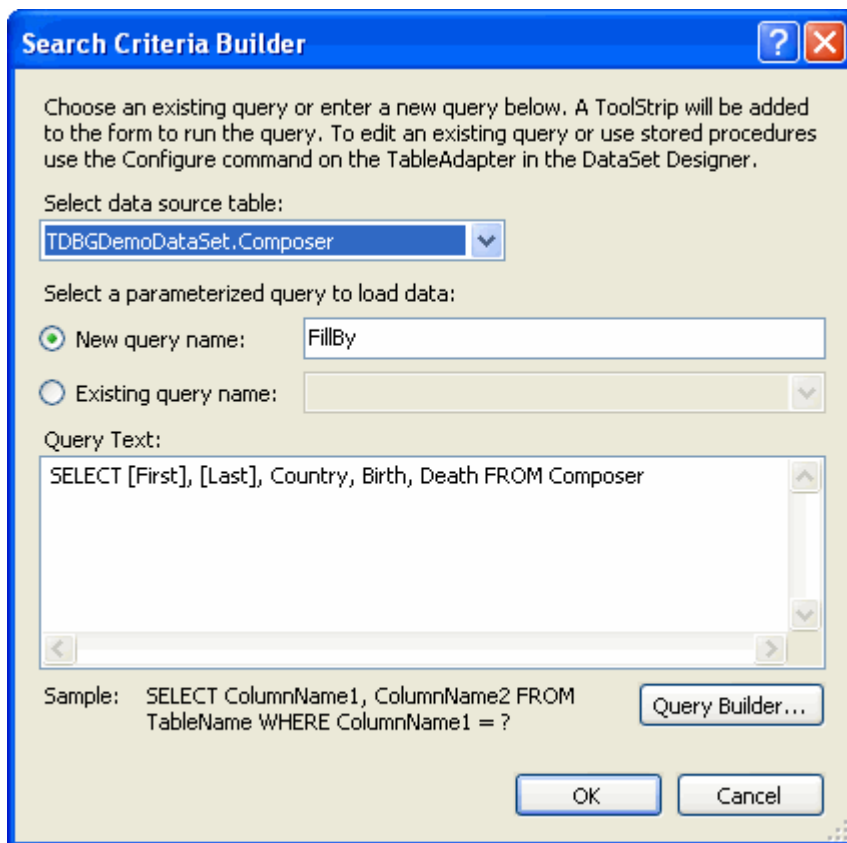
Dock in parent container/Undock in parent container

Clicking **Dock in parent container** sets the **Dock** property for [C1TrueDBGrid](#) to **Fill**.

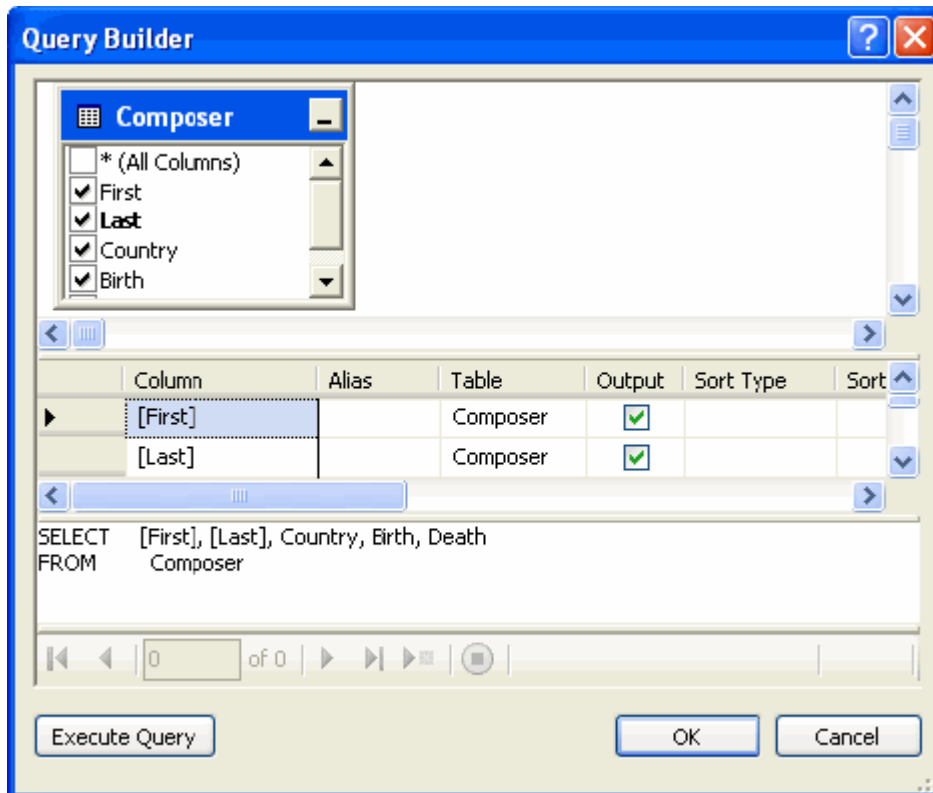
If [C1TrueDBGrid](#) is docked in the parent container, the option to undock **C1TrueDBGrid** from the parent container will be available. Clicking **Undock in parent container** sets the **Dock** property for [C1TrueDBGrid](#) to **None**.

Add Query (available only when bound to a data source)

Clicking **Add Query** opens the **Search Criteria Builder** dialog box, which allows you to create or modify a query.

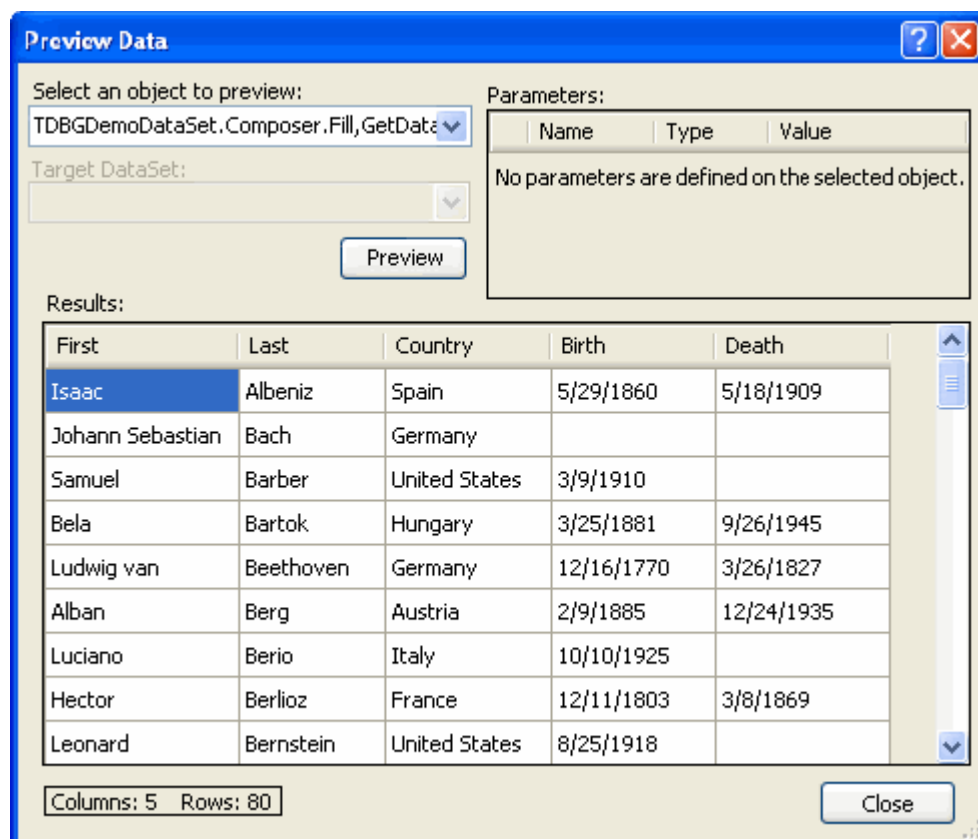


Instead of entering a query in the **Query Text** box, you can use the **Query Builder** to build a query by clicking on the **Query Builder** button.



Preview Data (available only when bound to a data source)

Clicking **Preview Data** opens the **Preview Data** dialog box, where you can preview the data in the DataSet.



Column Tasks Menu

The **Column Tasks** menu allows you to set the column caption, data field, input mask, aggregate, caption style, column style, and value items for a column, as well as set the following properties: [Visible](#), [ColumnVisible](#), and [EnableDateTimeEditor](#).

The **Column Tasks** menu can only be accessed when the grid is bound to a data source. To access the **Column Tasks** menu, either click a column in the grid or select **Column Tasks** from the **C1TrueDBGrid Tasks** menu.

The screenshot shows the 'C1TrueDBGrid Tasks' dialog box with the 'Column Tasks' tab selected. The dialog has a title bar 'C1TrueDBGrid Tasks' and a tab 'Column Tasks'. It contains several settings:

- Select Column:** A dropdown menu with 'First' selected.
- Column Caption:** A text box containing 'First'.
- Data Field:** A dropdown menu with 'First' selected.
- Input Mask:** A text box with an empty value and a button with three dots.
- Aggregate:** A dropdown menu with 'None' selected.
- Checkboxes:**
 - ☒ Visible
 - ☐ Visible when Grouped
 - ☒ Edit using DateTimePicker
- Buttons:**
 - Caption Style...
 - Column Style...
 - Value Items...
 - C1TrueDBGrid Tasks
 - Dock in parent container
 - Add Query...
 - Preview Data...

The **Column Tasks** menu operates as follows:

Select Column

Clicking the drop-down arrow in the **Select Column** box opens a list of available columns in the grid. If you clicked a column in the grid to open the tasks menu, that column will be the selected column.

C1TrueDBGrid Tasks

Column Tasks

Select Column: [First](#)

Column Caption: [First](#)

Data Field: [Country](#)

Input Mask: [None](#)

Aggregate: [None](#)

☒ Visible

☐ Visible when Grouped

☒ Edit using DateTimePicker

[Caption Style...](#)

[Column Style...](#)

[Value Items...](#)

[C1TrueDBGrid Tasks](#)

[Undock in parent container](#)

[Add Query...](#)

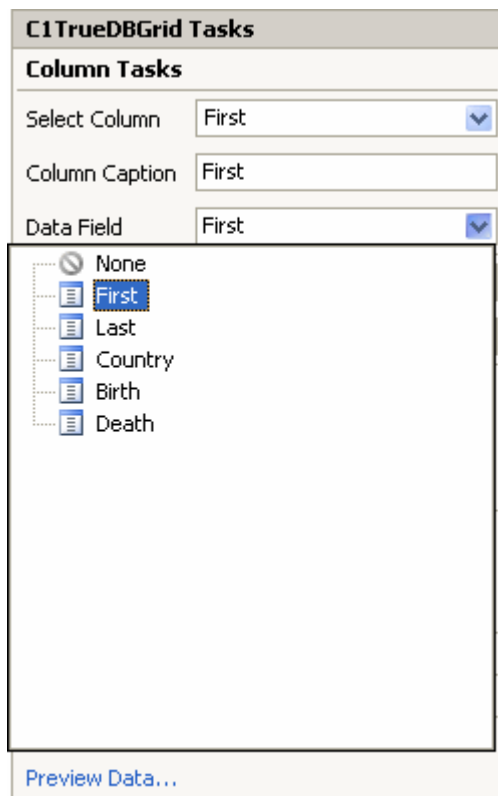
[Preview Data...](#)

Column Caption

Entering a caption into the **Column Caption** box set the [Caption](#) property for the column.

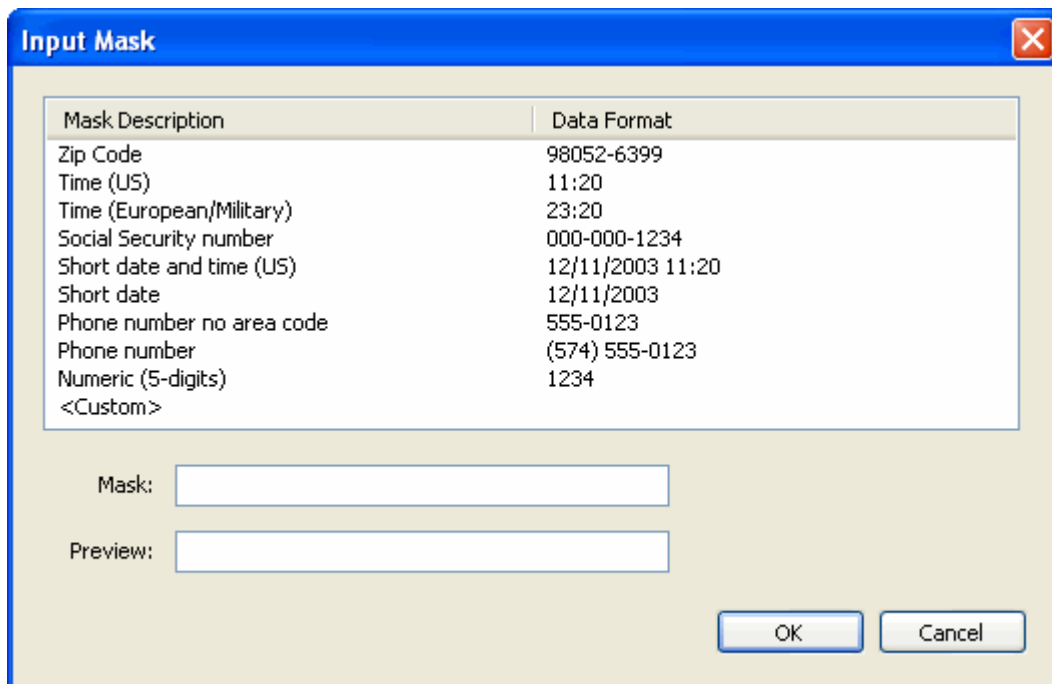
Data Field

Clicking the drop-down arrow in the **Data Field** box opens a list of available fields in the data source.



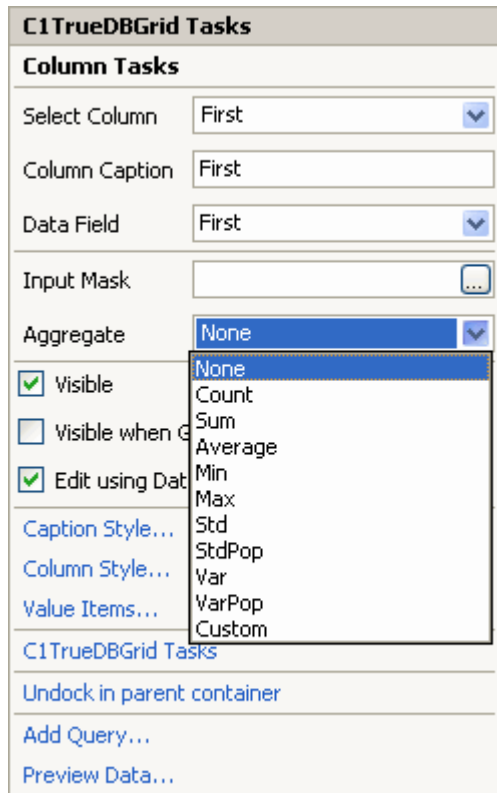
Input Mask

Clicking the **ellipsis** button in the **Input Mask** box opens the **Input Mask** dialog box.



Aggregate

Clicking the drop-down arrow in the **Aggregate** box opens a list of available aggregate functions, such as **Count**, **Sum**, **Average**, **Min**, **Max**, **Std**, **StdPop**, **Var**, **VarPop**, and **Custom**. For details on the available aggregate functions, see the [AggregateEnum](#) enumeration.



Visible

Selecting the **Visible** check box sets the [Visible](#) property to **True** for the selected column. The default is checked.

Visible when Grouped

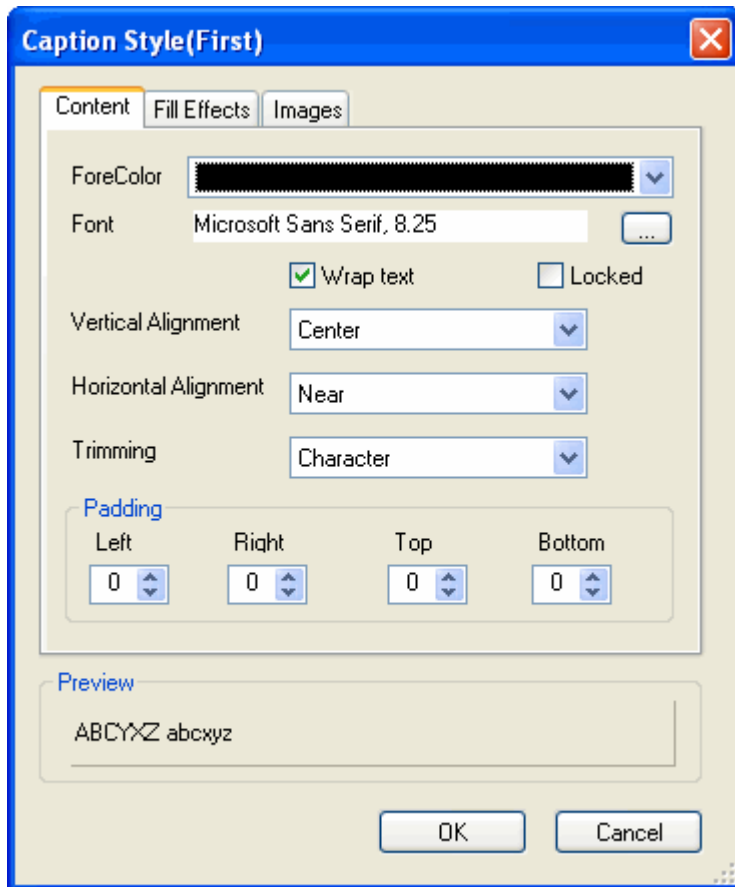
Selecting the **Visible when Grouped** check box sets the [ColumnVisible](#) to **True** for the selected column. The default is unchecked.

Edit using DateTimePicker

Selecting the **Edit using DateTimePicker** check box sets the [EnableDateTimeEditor](#) property to **True** for the selected column. The default is checked.

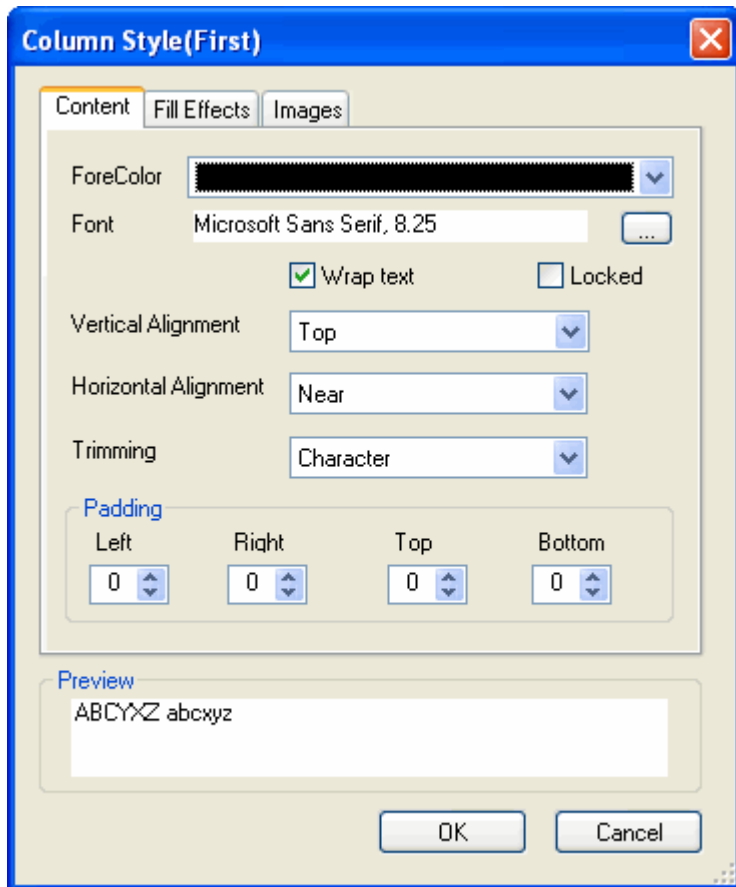
Caption Style

Clicking **Caption Style** opens the **Caption Style** editor for the selected column, which allows you to specify the properties for the caption, including style, fill effects, and images.



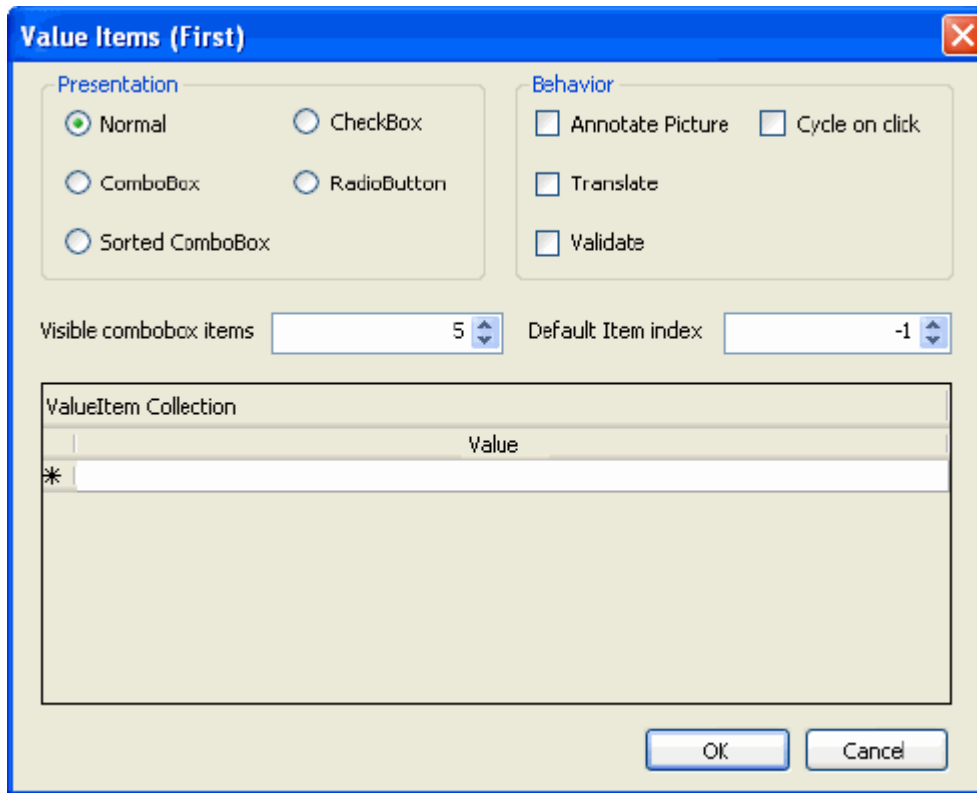
Column Style

Clicking **Column Style** opens the **Column Style** editor for the selected column, which allows you to specify properties for the column, including style, fill effects, and images.



Value Items

Clicking **Value Items** opens the **Value Items** editor for the selected column, which allows you to specify properties for the presentation and behavior of the **ValueItems** in the column.



C1TrueDBGrid Tasks

Clicking **C1TrueDBGrid Tasks** returns you to the **C1TrueDBGrid Tasks** menu. For details on the **C1TrueDBGrid Tasks** menu, see [C1TrueDBGrid Tasks Menu](#).

Dock in parent container

Clicking **Dock in parent container** sets the **Dock** property for [C1TrueDBGrid](#) to **Fill**.

If [C1TrueDBGrid](#) is docked in the parent container, the option to undock [C1TrueDBGrid](#) from the parent container will be available. Clicking **Undock in parent container** sets the **Dock** property for [C1TrueDBGrid](#) to **None**.

Add Query

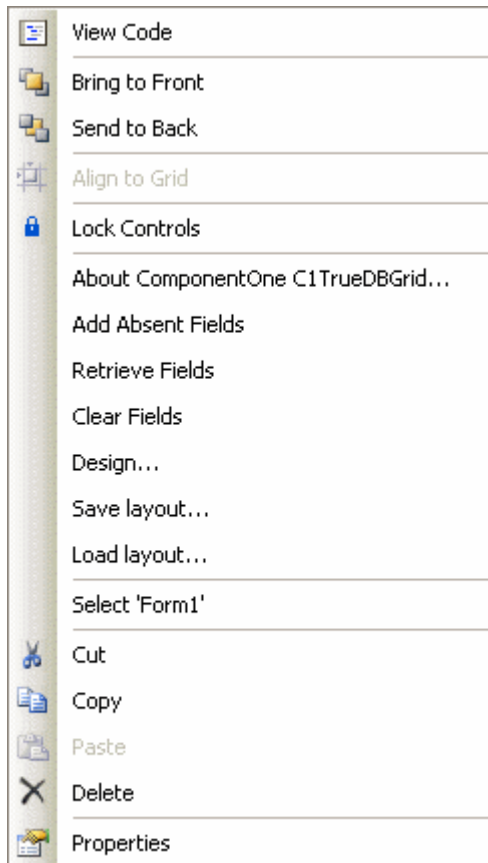
Clicking **Add Query** opens the **Search Criteria Builder** dialog box, which allows you to create or modify a query.

Preview Data

Clicking **Preview Data** opens the **Preview Data** dialog box, where you can preview the data in the DataSet.

C1TrueDBGrid Context Menu

Right-click anywhere on the grid to display the **True DBGrid for WinForms** context menu, which is a context menu that Visual Basic provides for all .NET controls. Although the **C1TrueDBGrid**'s context menu has a few extra features.



The context menu commands operate as follows:

About ComponentOne C1TrueDBGrid

This command displays the grid's **About** dialog box, which is helpful in finding the build number of the grid.

Add Absent Fields

This option adds fields from the data source that are not currently in the Columns collection.

Retrieve Fields/Clear Fields

These commands initiate the **RetrieveFields** and [ClearFields](#) methods of the grid. **RetrieveFields** goes back to the data source and retrieves all of the formatting information and base data for the column. **ClearFields** clears out any existing column formatting.

Design

This command brings up the **C1TrueDBGrid Designer**. This designer will enable the developer to add or delete columns, set **DataColumn**, **DisplayColumn**, and **Split** properties, and configure column order and many other aspects of the grid's design. For more information see [Using the C1TrueDBGrid Designer](#).

Save Layout/Load Layout

These commands save the current layout of the grid (style properties, column widths, and so on) to an XML file, or retrieve the XML file, loading a new grid setup.

Cut, Copy, Paste, Delete

These commands are identical to those on the Visual Studio **Edit** menu. **Cut** (CTRL+X) moves the grid from the Visual Basic form to the Clipboard. **Copy** (CTRL+C) moves a copy of the grid to the Clipboard while leaving the grid on the form intact. **Paste** (CTRL+V) copies the grid from the Clipboard to the form. **Delete** (the DEL key) removes the grid but does not move it to the Clipboard. You can undo the **Delete** command by selecting **Undo** (CTRL+Z) from the Visual Basic **Edit** menu.

Bring To Front, Send To Back

These commands control the z-order of the grid relative to the other objects on the Visual Basic form. **Bring To Front** places the grid in front of other objects; **Send To Back** places it behind other objects.

View Code

This command displays the grid's code window, which enables the viewing and editing of the grid's event handling code.

Align to Grid

This command automatically aligns the outer edges of the grid control to the design-time grid lines on the form.

Run-Time Interaction

The following topics describe how end users of your grid applications can interact with **True DBGrid for WinForms** at run time. You can give users the ability to perform any or all of the following:

- Navigate within the grid using the mouse or keyboard.
- Select rows or columns.
- Add, update, and delete records.
- Configure the grid's layout.

In the following sections, the properties and events associated with a particular user action are noted where applicable.

Navigation and Scrolling

The following sections describe the grid's default navigation and scrolling behavior. You always have complete control over the behavior of the TAB and arrow keys as well as the position of the current cell when a row or split boundary is reached.

Mouse Interaction

When the user clicks a non-current cell, the grid fires the [BeforeRowColChange](#) event. Unless this event is cancelled, the clicked cell becomes current and the grid subsequently fires the [RowColChange](#) event after any pending update operations have completed. The only exceptions to this are:

- If the user clicks a cell in a column or split that has the [AllowFocus](#) property set to **False**, and the cell belongs to the current row, then the current cell does not change.
- If the user clicks a cell in a column or split that has the [AllowFocus](#) property set to **False**, and the cell does not belong to the current row, then the current row changes, but the column with the focus retains it.
- If the current cell has been modified, and the [BeforeColUpdate](#) event is canceled, then the current cell does not change.
- If the current row has been modified, and the user clicks a cell in a different row, and the [BeforeUpdate](#) event is canceled, then the current cell does not change.

The user can also use the mouse to manipulate the grid's scroll bars, bringing cells that lie outside the grid's display area into view. The vertical scroll bar governs rows; the horizontal scroll bar governs columns. The [HScrollBar](#) property controls whether the horizontal scroll bars are displayed, while the [VScrollBar](#) property controls the vertical scroll bar.

Note that the scroll bars do not change the current cell. Therefore, the current cell may not always be visible.

To respond to vertical scrolling operations in code, use the [FirstRowChange](#) event. To respond to horizontal scrolling operations in code, use the [LeftColChange](#) event.

Clicking the Rightmost Column

The grid always displays the leftmost column (the first visible column) in its entirety. The rightmost column, however, is usually clipped. The behavior of the last partially visible column when clicked by the user is controlled by the grid's [ExposeCellMode](#) property.

The default value for the [ExposeCellMode](#) property is [ExposeCellModeEnum.ScrollOnSelect](#). If the user clicks the rightmost column when it is partially visible, the grid will scroll to the left to display this column in its entirety. This may be less desirable for users who commonly click on the grid to begin editing, as the grid will always shift to the left when the user clicks on a partially visible rightmost column.

If [ExposeCellMode](#) is set to [ExposeCellModeEnum.ScrollOnEdit](#), the grid will not scroll when the rightmost visible column is clicked. However, if the user attempts to edit the cell, then the grid will scroll to the left to display the column in its entirety. This is how Microsoft Excel works and is probably the most familiar setting to users.

If [ExposeCellMode](#) is set to [ExposeCellModeEnum.ScrollNever](#), the grid will not scroll to make the rightmost column visible, even if the user subsequently attempts to edit the cell. Note that editing may be difficult if only a small portion of the column is visible. The chief reason to use this setting is to ensure enough space is available for editing (or if editing is disallowed) and to prevent the grid from shifting accidentally.

Note that the [ExposeCellMode](#) property controls the behavior of the rightmost visible column only when the user clicks it with the mouse. If the rightmost column becomes visible by code (setting the grid's [Col](#) property) or by keyboard navigation, then the grid will always scroll to make it totally visible.

Keyboard Interaction

True DBGrid for WinForms includes several keyboard shortcuts that can improve users' run-time interaction experience.

In Grid View

By default, the user can navigate the grid with the arrow keys, the ENTER key, the TAB key, the Page Up and Page Down keys, and the HOME and END keys.

| Key | Action |
|--------------------|--|
| Up/Down Arrows | These keys move the current cell to adjacent rows. |
| Left/Right Arrows | If the AllowArrows property is True (the default), these keys move the current cell to adjacent columns. If the AllowArrows property is False , then these keys move focus from control to control and cannot be used to move between cells. |
| ENTER | By default, the ENTER key behaves in the same manner as the RIGHT ARROW key, by moving the current cell from left to right along the adjacent columns. The behavior for the ENTER key can be modified by using the DirectionAfterEnter property. When editing a cell, the ENTER key will save edits and then move to the next cell. |
| TAB | If the TabAction property is set to Control Navigation (the default), the TAB key moves focus to the next control on the form as determined by the tab order. If the TabAction property is set to ColumnNavigation or GridNavigation , the TAB key moves the current cell to the next column, while SHIFT+TAB moves to the previous column. The differences between column and grid navigation are discussed in the next section. |
| PAGE UP, PAGE DOWN | These keys scroll the grid up or down an entire page at a time. Unlike the vertical scroll bar, the PAGE Up and PAGE DOWN keys change the current row by the number of visible rows in the grid's display. When paging up, the current row becomes the first row in the display area. When paging down, the current row becomes the last row in the display area, including the AddNew row. The current column does not change. |
| HOME, END | These keys move the current cell to the first or last column. If necessary, the grid will scroll horizontally so that the current cell becomes visible. The current row does not change. If the current cell is being edited, HOME and END move the insertion point to the beginning or end of the cell's text. |
| F2 | Switch between Edit mode (with insertion point displayed) and Navigation mode in a datasheet. When working in a form or report, press ESC to leave Navigation mode. |

| Key | Action |
|--------------------|--|
| F4 | This key shows or hides a combo box. |
| ALT+DOWN ARROW | You can use this key combination to show a combo box, such as a C1TrueDBG |
| ALT + LEFT ARROW | Collapses all open child grids. |
| ALT+RIGHT ARROW | Expands a child grid. |
| DELETE | The DELETE button deletes the row. In edit mode, the DELETE key deletes the selected contents of a cell, deleting content to the right of the cursor. |
| SPACE BAR | The SPACE BAR key initiates button clicks and check box and radio button selection and de-selection. |
| CTRL+C | This key combination copies the selected content to the Clipboard |
| CTRL+V | This key combination pastes the contents of the Clipboard to the selected location. |
| CTRL+X | This key combination cuts the selected content and copies it to the Clipboard |
| BACKSPACE, CTRL+H | In Edit mode you can use the BACKSPACE key or the CTRL+H key combination to delete content from a cell, deleting content to the left of the cursor. |
| TAB | The TAB key ends cell editing mode, saving any content changes and leaving the current cell selected. |
| INSERT | In cell editing mode, the INSERT key changes the way text is entered in a cell. When the INSERT key is active, inputted text overwrites existing content. |
| CTRL+UP/DOWN ARROW | These key combinations allow you to navigate in a list box. The CTRL+UP ARROW key combination allows you to navigate up to the previous item in a list box. The CTRL+DOWN ARROW lets you navigate down to the next item. |

Navigation at Row Boundaries

At row boundaries, namely the first and last column, grid navigation depends on the [WrapCellPointer](#) property. The following explanation assumes that the [AllowArrows](#) property is **True**, and that the [TabAction](#) property is set to either **ColumnNavigation** or **GridNavigation**.

| Key | Action |
|-------------------|--|
| Left/Right Arrows | If the WrapCellPointer property is set to True , the current cell wraps across row boundaries. If the current cell is in the last column, the RIGHT ARROW key moves it to the first column of the next row. If the current cell is in the first column, the LEFT ARROW key moves it to the last column of the previous row. If the WrapCellPointer property is set to False (default), these keys cannot move the current cell at row boundaries. |
| TAB | If the TabAction property is ColumnNavigation , the cell pointer does not wrap to an adjacent row, and the WrapCellPointer property is ignored. If the current cell is in the last column, TAB moves focus to the next control in the tab order. If the current cell is in the first column, SHIFT+TAB moves focus to the previous control in the tab order. If the TabAction property is GridNavigation and WrapCellPointer is True , TAB and SHIFT+TAB move the current cell to the next or previous row. The current cell will not cross row boundaries if WrapCellPointer is False . |

Navigation at Split Boundaries

At split boundaries, grid navigation depends on the [TabAcrossSplits](#) property as follows:

| Key | Action |
|-------------------|--|
| Left/Right Arrows | If the TabAcrossSplits property is set to True , these keys move the current cell across split boundaries to the next or previous split. If the TabAcrossSplits property is set to False (default), the behavior of these keys at split boundaries will be the same as their behavior at row boundaries. Note that a split's AllowFocus property must be True in order for these keys to move the current cell to that split. |
| TAB | The TAB and SHIFT+TAB keys honor TabAcrossSplits as previously described for the arrow keys. |

Restricting Cell Navigation

The [BeforeRowColChange](#) event can be used to prevent the user from moving to a different cell, regardless of whether the current cell is modified. Set the **Cancel** argument to **True** to keep another cell from becoming current.

If the current cell has been modified, use the [BeforeColUpdate](#) event to examine its value before moving to another grid cell. If the value entered is invalid, set the **Cancel** argument to **True** to prevent the current cell from changing, and optionally beep or display an error message for the user. The [BeforeColUpdate](#) event provides a flexible way to validate user input and restrict cell navigation.

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_BeforeColUpdate(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.BeforeColUpdateEventArgs) Handles C1TrueDBGrid1.BeforeColUpdate
    Dim CharCode As Integer
    If e.ColIndex = 1 Then

        ' Data in Column 1 must start with upper case.
        CharCode = Asc(Me.C1TrueDBGrid1.Columns(1).Text)
        If CharCode > 64 And CharCode < 91 Then Exit Sub

        ' Display warning message for user.
        MessageBox.Show("Last name must start with upper case")

        ' Data validation fails, prohibit user from moving to another cell.
        e.Cancel = True
    End If
End Sub
```

To write code in C#

C#

```
private void c1TrueDBGrid1_BeforeColUpdate(object sender,
C1.Win.C1TrueDBGrid.BeforeColUpdateEventArgs e) {
    int CharCode;
    if ( e.ColIndex == 1 )
```

```
{
    // Data in Column 1 must start with upper case.
    CharCode = this.clTrueDBGrid1.Columns[1].Text[0];
    if ( CharCode > 64 && CharCode < 91 ) return;

    // Display warning message for user.
    MessageBox.Show("Last name must start with upper case");

    // Data validation fails, prohibit user from moving to another cell.
    e.Cancel = true;
}
}
```

Selection, Sorting, and Movement

The following sections describe how users can select columns, move selected columns, and select rows. You can restrict any or all of these operations at design time or in code.

Selecting Columns

If the [AllowColSelect](#) property is **True**, the user can select an individual column or a range of columns with the mouse. Nonadjacent column selections are not supported.

When the user points to the header of an unselected column, the mouse pointer changes to a down arrow to indicate that the column can be selected:

| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

When the user clicks a column header, that column is selected and highlighted, and any columns or rows that were previously selected are deselected:

| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

There are two ways for the user to select a range of columns:

- After selecting the first column in the range by clicking its header, the user can select the last column in the range by holding down the SHIFT key and clicking another column header. If necessary, the horizontal scroll bar can be used to bring additional columns into view.
- Alternatively, the user can hold and drag the mouse pointer within the column headers to select multiple columns.

In order to manipulate the columns that have been selected at run-time, query the [SelectedColumnCollection](#). This is a collection of all the [C1DataColumn](#) objects for the selected columns. For instance, if columns 5 through 10 are selected, the [SelectedColumnCollection](#) will have six members, each a [C1DataColumn](#) object. This feature enables the display properties of the column to be altered directly. Using the **Item** property to access the [C1DisplayColumn](#) properties, the code to change the foreground color to red for the first column selected would be:

To write code in Visual Basic

Visual Basic

```
Dim dc as C1TrueDBGrid.C1DataColumn

dc = Me.C1TrueDBGrid1.SelectedCols(0)
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(dc).Style.ForeColor =
System.Drawing.Color.Red
```

To write code in C#

C#

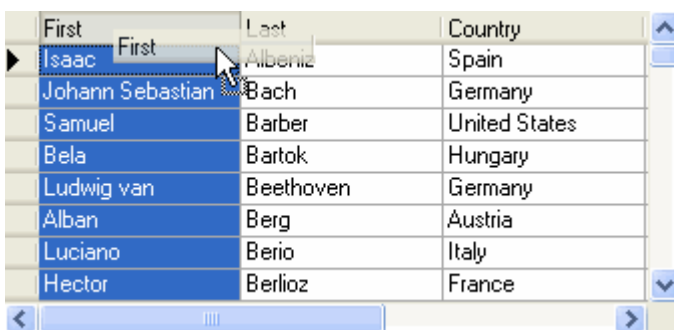
```
dc as C1TrueDBGrid.C1DataColumn;

dc = this.c1TrueDBGrid1.SelectedCols[0];
this.c1TrueDBGrid1.Splits[0].DisplayColumns[dc].Style.ForeColor =
System.Drawing.Color.Red;
```

Prevent a column selection from occurring at run time by setting the **Cancel** argument to **True** in the grid's [SelChange](#) event.

Moving Columns

If the [AllowColMove](#) property is **True**, the user can move previously selected columns as a unit to a different location by pressing the mouse button within the header area of any selected column. The pointer will change to an arrow with a column header box on its tip, a small box at its lower right corner, and a position marker consisting of two red triangles will appear at the left edge of the column being pointed to and highlighted.



The user specifies the desired location of the selected columns by dragging position marker, which changes position as the mouse pointer crosses the right edge of a column.

| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

The user completes the operation by releasing the mouse button, which moves the selected columns immediately to the left of the position marker. The moved columns remain selected.

| Last | First | Country |
|-----------|------------------|---------------|
| Albeniz | Isaac | Spain |
| Bach | Johann Sebastian | Germany |
| Barber | Samuel | United States |
| Bartok | Bela | Hungary |
| Beethoven | Ludwig van | Germany |
| Berg | Alban | Austria |
| Berio | Luciano | Italy |
| Berlioz | Hector | France |

If the user drags the marker to a position within the currently selected range, no movement occurs. Columns that are not selected cannot be moved interactively.

When a move occurs, the index in the Columns Collection is adjusted for all affected columns.

Prevent interactive column movement from occurring at run time by setting the **Cancel** argument to **True** in the [ColMove](#) event.

Moving Columns at Run Time

If the [AllowColMove](#) property is **True**, the user can move columns at run time also. Since there is no order property for a [C1DisplayColumn](#) the [C1DisplayColumnCollection](#) needs to be manipulated to move a column at run time. The [C1DisplayColumnCollection](#) holds all of the columns in a split. So to move a column, the user needs to remove the [C1DisplayColumn](#) from the collection, and then replace the column in the new position. The commonplace collection methods of **RemoveAt** and **Add** help accomplish this quite easily. The code which would transpose the first two columns in the default split would look as follows:

To write code in Visual Basic

Visual Basic

```
Dim dc as C1TrueDBGrid.C1DisplayColumn
dc = Me.C1TrueDBGrid1.Splits(0).DisplayColumns(1)
Me.C1TrueDBGrid1.Splits(0).DisplayColumns.RemoveAt(1)
Me.C1TrueDBGrid1.Splits(0).DisplayColumns.Insert(0, dc)
```

To write code in C#

C#

```
dc as C1TrueDBGrid.C1DisplayColumn;
dc = this.c1TrueDBGrid1.Splits(0).DisplayColumns[1];
this.c1TrueDBGrid1.Splits[0].DisplayColumns.RemoveAt(1);
this.c1TrueDBGrid1.Splits[0].DisplayColumns.Insert(0, dc);
```

Sorting Columns

If the [AllowSort](#) property is **True** (default), the user can sort columns by clicking on column headers. When the mouse is over a column header, the header will appear highlighted:

| First | Last | Country ↓ | |
|------------------|-----------|---------------|--|
| Isaac | Albeniz | Spain | |
| Johann Sebastian | Bach | Germany | |
| Samuel | Barber | United States | |
| Bela | Bartok | Hungary | |
| Ludwig van | Beethoven | Germany | |
| Alban | Berg | Austria | |
| Luciano | Berio | Italy | |
| Hector | Berlioz | France | |

If the user clicks on the column header once with the mouse, the column is sorted and a sort indicator arrow appears in the column header to indicate the direction of the sort:

| First | Last | Country / ↓ | |
|-----------------|-------------|-------------|--|
| Alban | Berg | Austria | |
| Anton | Bruckner | Austria | |
| Gustav | Mahler | Austria | |
| Wolfgang Amadeu | Mozart | Austria | |
| Arnold | Schoenberg | Austria | |
| Franz | Schubert | Austria | |
| Anton von | Webern | Austria | |
| Heitor | Villa-Lobos | Brazil | |

If the user clicks the header again, the sort is reversed and the direction of the sort indicator arrow is also reversed:

| First | Last | Country ▾ ↓ | |
|---------|-----------|---------------|--|
| Samuel | Barber | United States | |
| Leonard | Bernstein | United States | |
| Aaron | Copland | United States | |
| George | Gershwin | United States | |
| Charles | Ives | United States | |
| Virgil | Thomson | United States | |
| Ernest | Bloch | Switzerland | |
| Isaac | Albeniz | Spain | |

If the [AllowColSelect](#) property is set to **False** the column will not be selected when sorting.

To prevent users from sorting columns at run time set the [AllowSort](#) property to **False**, for more information see the

[Disabling Column Sorting](#) topic.

Selecting Rows

Row selection is also very easy to master with **True DBGrid for WinForms**. When the cursor hovers over the record selector to the left of a row, a small arrow will indicate that this row is about to be selected. Then by clicking on this record selector, the row then becomes selected. To select a contiguous range of rows, manipulate the arrow keys while holding down the shift button. This will select the cells in every field for the range of rows that the user has selected. To select a non-contiguous range of rows, click on the rows to be selected while holding down the CTRL button. This will select the cells in every field for the set of non-contiguous rows that the user has selected.

In order to find out which rows have been selected at run-time, query the [SelectedRowCollection](#). This is a collection of all the row indices for the selected rows. For instance, if rows 5 through 10 are selected, the [SelectedRowCollection](#) will have six members. Each member will be an integer value and will be an integer that corresponds to the absolute row position of the selected row.

Selecting a Range of Cells

When it comes to cell selection, **True DBGrid for WinForms** has multi-select capability enabled at all times, very similar to Microsoft Excel. By clicking on a cell and dragging the mouse, or by using the arrow keys, a range of cells can be selected. In turn, by clicking on a record selector and manipulating the mouse a set of rows can be selected. This range is not restricted to the row or column of the initial cells origin, although non-contiguous cell selection is not supported.

When a range of cells is selected the grid's [SelRange](#) property becomes **True**. This will indicate that neither just the [SelectedRowCollection](#) nor just the [SelectedColumnCollection](#) collections will tell which cells are selected. By evaluating both collections, though, and taking the intersection of the two collections, the selected cell range can be discovered at run time. For instance, if the user clicked on the second row, second column, and dragged the mouse to the fourth row, fourth column, the [SelectedRowCollection](#) collection would contain the integers 2, 3, and 4, while the [SelectedColumnCollection](#) collection would contain the [C1DataColumn](#) objects for columns 2, 3, and 4. From this, it can be discerned at run-time that there is a nine-cell range selected from column 2, row 2, to column 4, row 4.

Sizing and Splitting

The following sections describe how users can resize rows, columns, and splits. Restrict any or all of these operations at design time or in code.

Sizing Rows

If the [AllowRowSizing](#) property is set to either **RowSizingEnum.AllRows** or **RowSizingEnum.IndividualRows**, the user can change the row height at run time. When the user points to a row divider in the record selector column, the pointer changes to a vertical double arrow that the user can drag to adjust the height of all rows.

| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

Dragging the pointer upward makes the rows smaller; dragging it downward makes the rows larger. If the property is set to **AllRows**, then all rows in the grid will be resized to the same height; it is not possible to resize individual rows. If the property is set to **IndividualRows**, then each row can be sized independently.

In addition, if the grid does not display the record selector column (that is, the [RecordSelectors](#) property is **False**), users cannot interactively change the row height.

The [RowHeight](#) property of the grid will be adjusted when the user completes the resize operation.

Prevent row resizing from occurring at run time by setting the **Cancel** argument to **True** in the [RowResize](#) event. Change the **RowHeight** of the grid in code, even if [AllowRowSizing](#) is [RowSizingEnum.None](#) or the **RowResize** event is cancelled.

Sizing Columns

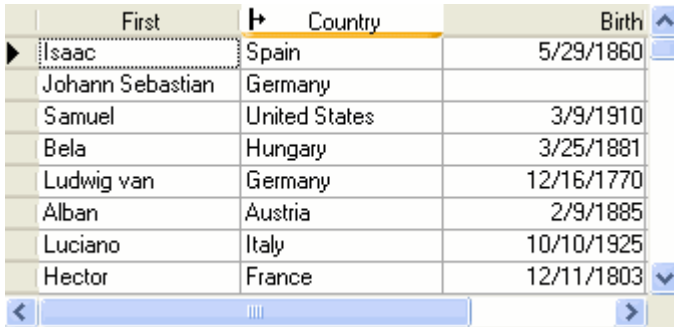
If the [AllowSizing](#) property is **True** for a column, the user can adjust the width of the column individually at run time. When the user points to the divider at the right edge of a column's header, the pointer changes to a horizontal double arrow that the user can drag to resize the column in question.

| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

Dragging the pointer to the left makes the column smaller; dragging it to the right makes the column larger. The column's [Width](#) property will be adjusted when the user completes the resize operation.

If the grid does not display column headers (that is, the [ColumnHeaders](#) property is **False**), the horizontal double arrow will appear when the pointer is over the column divider within the grid's data area.

If the user drags the pointer all the way to the left, the column retains its original **Width** property setting, but its [Visible](#) property is set to **False**. To make the column visible again, the user can point to the right side of the divider of the column that preceded it. The pointer turns into a vertical bar with a right arrow.



| First | Country | Birth |
|------------------|---------------|------------|
| Isaac | Spain | 5/29/1860 |
| Johann Sebastian | Germany | |
| Samuel | United States | 3/9/1910 |
| Bela | Hungary | 3/25/1881 |
| Ludwig van | Germany | 12/16/1770 |
| Alban | Austria | 2/9/1885 |
| Luciano | Italy | 10/10/1925 |
| Hector | France | 12/11/1803 |

Dragging the pointer to the right establishes a new column width and sets the column's **Visible** property back to **True**.

Another way to resize columns is to use the [AutoSize](#) method to specify auto-sizing for a specific **Column** object in code. When a column is auto-sized, its width is adjusted to fit the longest *visible* field in that column. Longer records that are not displayed in a visible row when **AutoSize** is invoked do not participate in the width calculations. Furthermore, if the specified column is either hidden or scrolled out of view, a trappable error occurs.

Prevent column resizing from occurring at run time by setting the **Cancel** argument to **True** in the [ColResize](#) event. Change the width of a column in code, even if [AllowSizing](#) is **False** for that column.

Database Operations

The editing, deleting, and adding permissions granted to the user at run time are controlled by the [AllowUpdate](#), [AllowDelete](#), and [AllowAddNew](#) properties. The default values of these properties are:

| Property | Default |
|-------------|---------|
| AllowUpdate | True |
| AllowDelete | False |
| AllowAddNew | False |

Note that these properties only control user interaction with the grid at run time. They do not control whether database operations can be performed by the DataSet or other bound controls, or by the application code.

Editing Data

True DBGrid for WinForms' [AllowUpdate](#) property must be **True** in order for the user to edit data in the grid. The default value is **True**.

If the user moves to a cell and starts typing, the cell's data will be replaced by what is typed. Alternatively, clicking within the current cell will put the grid into edit mode (its [EditActive](#) property becomes **True**), enabling the user to modify the cell's data.

While editing, the LEFT ARROW and RIGHT ARROW keys move the insertion point within the cell. If the insertion point is at the beginning or end of the cell's text, the LEFT ARROW and RIGHT ARROW keys will terminate editing by moving to the adjacent cell. The UP ARROW and DOWN ARROW keys terminate editing by moving the current cell to the row above or below the current one. The user can also end editing without moving the current cell by pressing the ENTER key.

When one or more cells in a row have been modified, a pencil icon will appear in the record selector column to indicate that data in the row has been changed. The pencil icon does **not** mean that the grid's **EditActive** property is **True**; it means that the grid's [DataChanged](#) property is **True**. To cancel the changes made to the current cell, the user can press the ESC key. In fact, before moving to another row, the user can revisit any column within the current row

and press the ESC key to restore the cell to its original value. If the user repeats this procedure for all modified cells in the row, the pencil icon in the record selector will disappear.

Moving to another row by clicking it, using the UP ARROW or DOWN ARROW keys, or by clicking the navigation buttons of the Data control will update the modified record to the database. If the update is successful, the pencil icon will disappear. If no grid columns have been modified, no update will occur when changing rows.

Adding a New Record

True DBGrid for WinForms' [AllowAddNew](#) property must be **True** in order for the user to add new records to the grid interactively. The default value is **False**.

If the [AllowAddNew](#) property is **True**, an empty *AddNew row*, marked by an asterisk in the record selector column, will be displayed after the last record. The user can initiate an add operation by navigating to the AddNew row, either by clicking it or by using the DOWN ARROW key, then typing new data. The first character typed will cause the grid to insert a blank row before the AddNew row. The newly inserted blank row becomes the current row, and the grid fires the [OnAddNew](#) event.

At this point, the new row exists only in the grid—it does not have a bookmark, and it does not yet represent a physical database record. The new row is added to the underlying data source when the user navigates to another data row or the AddNew row.

Deleting a Record

True DBGrid for WinForms' [AllowDelete](#) property must be **True** in order for the user to delete records through the grid. The default value is **False**.

To delete a record, the user selects the row to be deleted by clicking its record selector, then pressing the DEL key. Only one record can be deleted at a time. The user cannot select multiple records and press the DEL key to delete them all.

In order for the record to be deleted, the grid must have focus so it can receive the DEL key. Clicking the grid's record selector column does **not** set focus to the grid. However, if you always want the grid to receive focus when the user clicks the record selector column, set focus to the grid in the grid's [SelChange](#) event:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_SelChange(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.CancelEventArgs) Handles C1TrueDBGrid1.SelChange
    Me.C1TrueDBGrid1.Focus()
End Sub
```

To write code in C#

C#

```
private void C1TrueDBGrid1_SelChange(object sender,
C1.Win.C1TrueDBGrid.CancelEventArgs e)
{
    this.c1TrueDBGrid1.Focus();
}
```

Customized Grid Editors

The following sections describe how to use and create custom grid editors.

Using Custom Editors

The built-in editors provide a lot of flexibility and power, but in some cases you may want to use external controls as specialized editors. For example, you may want to use the **C1NumericEdit** control that provides a drop-down calculator for entering numbers, or an editor for selecting from multi-column lists, or a specialized control that you wrote to edit your business objects.



Note: The **C1NumericEdit** control is one of the **Input for WinForms** controls. For more information on the **C1NumericEdit** control, please refer to the **Input for WinForms** documentation which is available on [ComponentOne HelpCentral](#).

Any control that derives from the base **Control** class can be used as a basic grid editor. Controls that implement the **IC1EmbeddedEditor** interface can provide better integration with the grid and more advanced features. For details on the **IC1EmbeddedEditor** interface, see the [Editor](#) property.

To use a control as a custom editor, all you have to do is associate an instance of the control with a grid column using the **Editor** property. You can do this in code using the **Editor** property. After that, the control will be automatically used by the grid.

For example, to use a **C1NumericEdit** control as a grid editor, follow these steps:

1. Add a [C1TrueDBGrid](#) control and a **C1NumericInput** control to the form.
2. For the **C1NumericInput** control, set the **BorderStyle** property to **None** and the **Visible** property to **False** either in the Properties window or by adding the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' Set up the custom editor.
Me.C1NumericEdit1.BorderStyle = BorderStyle.None
Me.C1NumericEdit1.Visible = False
```

To write code in C#

C#

```
// Set up the custom editor.
this.c1NumericEdit1.BorderStyle = BorderStyle.None;
this.c1NumericEdit1.Visible = false;
```

3. In the **Form_Load** event assign the custom editor to the grid column.

To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Assign the custom editor to the grid.
    Me.C1TrueDBGrid1.Columns(0).Editor = Me.C1NumericEdit1
```

```
End Sub
```

To write code in C#

```
C#  
  
private void Form_Load(object sender, EventArgs e)  
{  
  
    // Assign the custom editor to the grid.  
    this.c1TrueDBGrid1.Columns[0].Editor = this.c1NumericEdit1;  
}
```

Run the project and edit some values in the first column. Notice how the grid positions and initializes the **C1NumericEdit** control so you can edit cell values. When you are done editing a cell, click a different cell or press the TAB key to move to the next one. Notice how the new value is applied to the cell.

Creating Custom Editors

Any control that derives from the **Control** base class can be used as a grid editor. This is possible because the grid knows enough about the base class to access properties such as **Text** and **Bounds**, and events such as **Leave** and **TextChanged**. In many cases this level of support is adequate.

In some cases, however, you may want to use controls that do not follow the base class that closely. For example, a **DateTimePicker** control has a **Value** property that should be used to retrieve the edited value instead of **Text**. In these cases, you can implement one or more methods in the **IC1EmbeddedEditor** interface to override the default behavior. For example, all controls in the C1Input library support **IC1EmbeddedEditor** and therefore integrate closely with **C1TrueDBGrid** (and also **C1FlexGrid**).

The **IC1EmbeddedEditor** interface is fairly simple, and because the grid binds to it using late binding, you do not even have to implement all its members. Only implement the ones that make sense to your editor control.

The interface does provide enough flexibility to allow virtually any control to be used as a grid editor. You can even use **UITypeEditor** classes as grid editors. To do this, you need a wrapper class that:

1. Derives from **Control** (**UITypeEditor** does not).
2. Implements the **IC1EmbeddedEditor** interface.
3. Encapsulates the appropriate **UITypeEditor**.



Note: For a complete sample using this wrapper class, see the **CustomEditors** sample installed with **Winforms Edition**.

Using the **UITypeEditor** wrapper class, you can use any **UITypeEditors** with the **C1TrueDBGrid**. .NET provides several **UITypeEditors** for editing colors, fonts, file names, and so on. You can also write your own **UITypeEditors**, in some cases that is easier than writing a control.

Additional User Interaction Features

True DBGrid for WinForms provides additional data presentation and manipulation functionality that can be exposed to users at run time. For more information, please see the following topics:

- [Context-sensitive Help with CellTips](#)
- [Scroll Tracking and ScrollTips](#)
- [Hierarchical Data Display](#)
- [Drop-down Hierarchical Data Display](#)

- [Column Grouping](#)

Data Binding

The following topics describe how to bind to a data source, create and use unbound columns, and display data without binding to a data source.

Binding True DBGrid for WinForms to a Data Source

With an amazing ease of use, **True DBGrid for WinForms** can universally bind to any .NET data source. Requiring little or no code at all, the **C1TrueDBGrid** control can create a fully-navigational database browser in mere seconds.

True DBGrid for WinForms fully supports data binding to ADO.NET objects such as **DataTable**, **DataView** and **DataSet** objects. You also have an even easier option of binding to **ComponentOne DataObjects Express** data sources, **C1ExpressTable** and **C1ExpressConnection**. **C1TrueDBGrid** also fully supports the powerful **DataObjects for WinForms** framework included in the ComponentOne Studio Enterprise.

To associate a **True DBGrid for WinForms** control with an ADO.NET or **DataObjects for WinForms** data source, set the **DataSource** property of the grid to a **DataSet** on the same form. If the **DataSet** contains multiple tables, you can select a table name in the **DataMember** property combo box. The **DataSource** and **DataMember** properties can be set both through code, and through Visual Studio's Properties window. This is all that is required to make **True DBGrid for WinForms** fully aware of the database or **DataTable** in your application.

Once such a link exists, **True DBGrid for WinForms** and the **DataSet** automatically notify and respond to operations performed by the other, simplifying your application development.

Preserving the Grid's Layout

You can use the **SetDataBinding** method to bind the grid at run time. For example, the following code binds the **C1TrueDBGrid** control to the **Customers** table in the **DSCustomers** data source:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.SetDataBinding(Me.DsCustomers.Customers, "")
```

To write code in C#

C#

```
this.C1TrueDBGrid1.SetDataBinding(this.DsCustomers.Customers, "");
```

If the **DataSource** is reset through code, it will show all of the data in the grid and will not keep the initial layout created with the Designer. You can ensure that the grid layout remains as designed by using the **SetDataBinding** method with the **HoldFields** parameter set to **True**. For example

To write code in Visual Basic

Visual Basic

```
C1TrueDBGrid1.SetDataBinding(Me.DsCustomers.Customers, "", True)
```

To write code in C#

C#

```
this.c1TrueDBGrid1.SetDataBinding(this.DsCustomers.Customers, "", true);
```

For another example of using the **SetDataBinding(Object, String, Boolean)** method, see [Tutorial 2: Using True DBGrid for WinForms with SQL Query Results](#).

Note that you can create an unbound grid by using the SetDataBinding method without arguments. See [Creating an Unbound Grid](#) for details.

Using Unbound Columns

Normally, **True DBGrid for WinForms** automatically displays data from bound database fields. However, you may need to augment the set of fields present in your layouts with columns derived from database fields, or columns that are unrelated (or only loosely related) to database information. For example, if your database contains a *Balance* field, you may instead want to display two columns, *Credit* and *Debit*, to show positive and negative numbers separately. Or, you may want to look up data in another database, or convert field data to some other form, such as mapping numeric codes to textual descriptions.

To accomplish such tasks you can use unbound columns. The term *unbound column* refers to a column that is part of a *bound grid*, but is not tied directly to a database field.

Columns that do not have the [DataField](#) property set (that is, the DataField property is equal to an empty string), but do have the column [Caption](#) property set are considered unbound columns. The grid will request data for these columns through the [UnboundColumnFetch](#) event.

Columns with their DataField property set will be bound, if the DataField property is the same as one of the fields of the Data Source.

Columns with their DataField property set to a value that is not in the DataSet are ignored for the purposes of fetching data. Similarly, columns that have no value for both the DataField and [Caption](#) properties set are also ignored.

Creating Unbound Columns

The first step in using an unbound column is creating the column itself. This may be done in the designer by adding a column through the **C1TrueDBGrid Designer**. In code, unbound columns may be added using the [Insert](#) method of the C1DataColumnCollection. The column must be given a name by setting its [Caption](#) property. In the designer, this is done using the **C1TrueDBGrid Designer**. In code, the [Caption](#) property of the appropriate C1DataColumn object is set. C1DataColumn objects that are added to the C1DataColumnCollection cause a corresponding C1DisplayColumn to be added to the C1DisplayColumnCollection for all splits. The default visible property of the newly added C1DisplayColumn will be **False**.

When attempting to insert an unbound column in code, use the [Rebind](#) method to ensure that the column appears at the desired position within the grid:

To write code in Visual Basic

Visual Basic

```
Dim Col As New C1.Win.C1TrueDBGrid.C1DataColumn
Dim dc As C1.Win.C1TrueDBGrid.C1DisplayColumn

With Me.C1TrueDBGrid1
    .Columns.Insert(0, Col)
    Col.Caption = "Unbound"
    dc = .Splits(0).DisplayColumns.Item("Unbound")

    ' Move the newly added column to leftmost position in the grid.
    .Splits(0).DisplayColumns.RemoveAt(.Splits(0).DisplayColumns.IndexOf(dc))
    .Splits(0).DisplayColumns.Insert(0, dc)
    dc.Visible = True
    .Rebind(True)
```

End With

To write code in C#

C#

```
C1.Win.C1TrueDBGrid.C1DataColumn Col = new C1.Win.C1TrueDBGrid.C1DataColumn();
C1.Win.C1TrueDBGrid.C1DisplayColumn dc;
c1TrueDBGrid1.Columns.Insert(0, Col);
Col.Caption = "Unbound";
dc = c1TrueDBGrid1.Splits[0].DisplayColumns["Unbound"];

// Move the newly added column to leftmost position in the grid.
c1TrueDBGrid1.Splits[0].DisplayColumns.RemoveAt(C1TrueDBGrid1.Splits[0].DisplayColumns.IndexOf(dc));
c1TrueDBGrid1.Splits[0].DisplayColumns.Insert(0, dc);
dc.Visible = true;
c1TrueDBGrid1.Rebind(true);
```

When the grid needs to display the value of an unbound column, it fires the [UnboundColumnFetch](#) event. This event supplies the user with a row and column index as the means of identifying the grid cell being requested. The [Value](#) property to the event is of type Object that by default is **Null**, but can be changed to any desired value, and will be used to fill the contents of the cell specified by the given row and column index.

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_UnboundColumnFetch(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.UnboundColumnFetchEventArgs) Handles C1TrueDBGrid1.UnboundColumnFetch
```

To write code in C#

C#

```
private void c1TrueDBGrid1_UnboundColumnFetch(object sender,
C1.Win.C1TrueDBGrid.UnboundColumnFetchEventArgs e)
```

Implementing Multiple Unbound Columns

So far, our examples have demonstrated the [UnboundColumnFetch](#) event using only a single unbound column but more than one unbound column can be used. Since the [UnboundColumnFetch](#) is fired for each unbound column of each row, only one column value may be set at a time, and each column must be identified for the value to be properly determined. The second [UnboundColumnFetch](#) property, [Column](#), is used to identify the column of the grid for which the value is required.

To write code in Visual Basic

Visual Basic

```
' Will be used as the copy.
Dim dtCopy As Data.DataTable

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    dtCopy = Me.DataSet11.Tables(0).Copy()
End Sub

Private Sub C1TrueDBGrid1_UnboundColumnFetch(ByVal sender As System.Object, ByVal e
As C1.Win.C1TrueDBGrid.UnboundColumnFetchEventArgs) Handles
```

```

C1TrueDBGrid1.UnboundColumnFetch
    Select Case e.Column.Caption
    Case "Area"

        ' Calculate the "Area" column of the grid.
        e.Value = dtCopy.Rows(e.Row).Item("Length") *
dtCopy.Rows(e.Row).Item("Width")
    Case "Perimeter"

        ' Calculate the "Perimeter" column of the grid.
        e.Value = 2 * (dtCopy.Rows(e.Row).Item("Length") +
dtCopy.Rows(e.Row).Item("Width"))
    End Select
End Sub

```

To write code in C#

```

C#

// Will be used as the copy.
Data.DataTable dtCopy;

private void Form1_Load( System.Object sender, System.EventArgs e)
{
    dtCopy = this.DataSet11.Tables[0].Copy();
}

private void C1TrueDBGrid1_UnboundColumnFetch(object sender,
C1.Win.C1TrueDBGrid.UnboundColumnFetchEventArgs e)
{
    switch (e.Column.Caption;)
    {
        case "Area";

            // Calculate the "Area" column of the grid.
            e.value = dtCopy.Rows[e.Row].Item["Length"] *
dtCopy.Rows[e.Row].Item["Width"];
            break;
        case "Perimeter";

            // Calculate the "Perimeter" column of the grid.
            e.value = 2 * (dtCopy.Rows[e.Row].Item["Length"] +
dtCopy.Rows[e.Row].Item["Width"]);
            break;
    }
}

```

Updating Unbound Columns

In most cases, unbound columns will be read-only, as the values are derived from other data in the grid. In these cases, set the [Locked](#) property of the column's style to **True**.

If **Locked** is **False** and updates are allowed, the user can edit the values of an unbound column. If editing of an unbound column occurs, the row will be marked as dirty (a pencil icon will be shown in the record selector column) and the update sequence will be performed as usual. However, the grid does not know what to do with the modified data, since there is no database field in which to store it. In this situation the **UnboundColumnUpdated** event will be raised.

The **BeforeUpdate** event can be used to cancel the update operation. Therefore, if the unbound column is to be used in cooperation with another database, the update of the unbound column should be performed in **BeforeUpdate**. If the operation fails, then the event should be canceled. However, if the operation succeeds, then the bound update should be allowed to proceed. The bound update may then fail; hence any database actions associated with unbound columns would best be handled on a transactional basis.

If the bound update succeeds, the **AfterUpdate** event is fired, and the unbound column transaction should be committed. If the bound update fails, the unbound column transaction should be rolled back within .NET's trappable error handler, depending on how the update was initiated. If transactions are not available, then store the original unbound column values prior to the update, then perform another update to restore these values should the bound update fail.

Editing Unbound Columns

Another technique for updating an unbound column is to use the **AfterColUpdate** event to adjust the value of other (bound) columns. For example, imagine a pair of columns for *Debit* and *Credit*, as shown in this portion of a grid display:

| Credit | Debit |
|-------------|------------|
| \$13,677.13 | |
| \$3,288.50 | |
| \$5,466.78 | |
| | \$2,208.00 |
| | \$1,513.00 |
| \$297.30 | |
| \$2,344.50 | |

Assume that there is no database field for these, but that they are unbound columns that derive their value from a single *Balance* column, which is either positive or negative. From the user's perspective, it would be desirable to edit these values directly. From your perspective, it would be desirable to have the grid update the dependent *Balance* column automatically.

True DBGrid for WinForms makes such tasks easy. The following code would be put in the grid's **AfterColUpdate** event to cause either column to change the *Balance* column when updated:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_AfterColUpdate(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.ColEventArgs) Handles C1TrueDBGrid1.AfterColUpdate
    Dim row as Integer = Me.C1TrueDBGrid1.Row
    Me.C1TrueDBGrid1(row, "Balance") = -e.Column.DataColumn.Value
End Sub
```

To write code in C#

C#

```
private void ClTrueDBGrid1_AfterColUpdate(object sender,
Cl.Win.ClTrueDBGrid.ColEventArgs e)
{
    int row = this.clTrueDBGrid1.Row;
    this.clTrueDBGrid1[row, "Balance"] = -e.Column.DataColumn.Value;
}
```

Notice that, when updating these columns, the code actually changes the value of the *Balance* column, which is both bound and invisible.

Creating an Unbound Grid

True DBGrid for WinForms can display data without being bound to a *DataSource*. Creating an unbound grid can be done in a few steps.

To create an unbound grid, complete the following:

1. Begin by creating your columns. This can be done either in the designer or in code. For more information on creating columns, see [Creating Unbound Columns](#).

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.Columns.Add(New Cl.Win.ClTrueDBGrid.ClDataColumn("FirstName",
GetType(String)))
Me.ClTrueDBGrid1.Columns.Add(New Cl.Win.ClTrueDBGrid.ClDataColumn("LastName",
GetType(String)))
Me.ClTrueDBGrid1.Columns.Add(New Cl.Win.ClTrueDbGrid.ClDataColumn("DateOfBirth",
GetType(DateTime)))
```

To write code in C#

C#

```
this.clTrueDBGrid1.Columns.Add(new
Cl.Win.ClTrueDBGrid.ClDataColumn("FirstName",typeof(string)));
this.clTrueDBGrid1.Columns.Add(new
Cl.Win.ClTrueDBGrid.ClDataColumn("LateName",typeof(string)));
this.clTrueDBGrid1.Columns.Add(new
Cl.Win.ClTrueDBGrid.ClDataColumn("DateOfBirth",typeof(DateTime)));
```

2. Call the [SetDataBinding](#) method with no arguments.

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.SetDataBinding()
```

To write code in C#

C#

```
this.clTrueDBGrid1.SetDataBinding();
```

3. Use the [AddRow](#) or [AddRows](#) method to populate the grid.

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.AddRow("John;Doe;11/29/1985")
Me.C1TrueDBGrid1.AddRow("Jane;Doe;7/12/1980")

Dim index As Integer = Me.C1TrueDBGrid1.AddRows(2)
Dim i As Integer
For i = index To 1
    Me.C1TrueDBGrid1(i, "FirstName") = "Joe"
    Me.C1TrueDBGrid1(i, "LastName") = "Doe"
    Me.C1TrueDBGrid1(i, "DateOfBirth") = New DateTime(2000, 1, 15)
Next i
```

To write code in C#

C#

```
this.c1TrueDBGrid1.AddRow("John;Doe;11/29/1985");
this.c1TrueDBGrid1.AddRow("Jane;Doe;7/12/1980");

int index = this.c1TrueDBGrid1.AddRows(2);
for(int i=index; i < 2; i++)
{
    this.c1TrueDBGrid1[i,"FirstName"] = "Joe";
    this.c1TrueDBGrid1[i, "LastName"] = "Doe";
    this.c1TrueDBGrid1[i, "DateOfBirth"] = new DateTime(2000,1, 15);
}
```

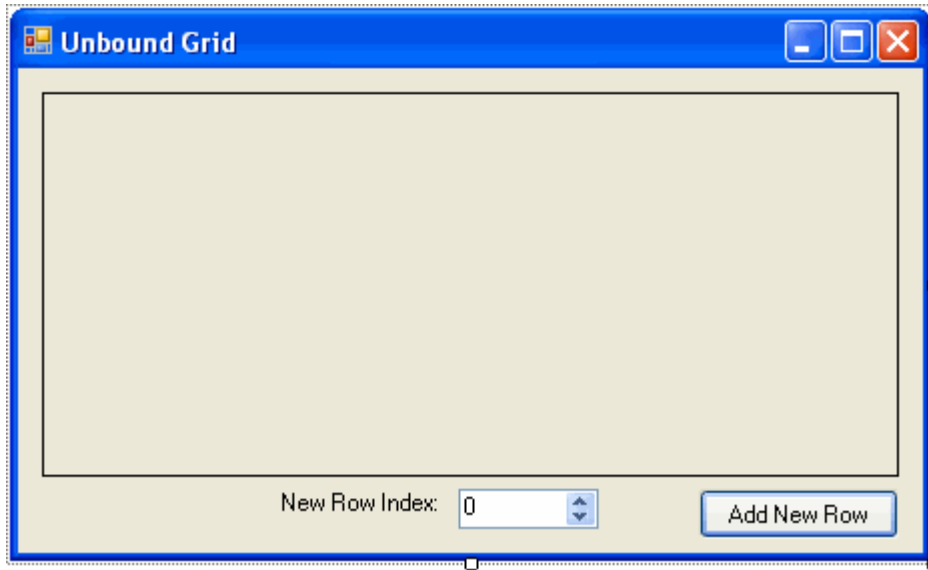
You have successfully created an unbound grid.

Adding New Rows to an Unbound Grid

You can now easily add new rows to an unbound grid by using the **C1TrueDBGrid.NewRow** method which creates a new `System.Data.DataRow` with the same schema as the unbound grid. In the following steps you'll use the **C1TrueDBGrid.Rows** collection, which gets the `DataRowCollection` for an unbound grid, and the **C1TrueDBGrid.NewRow** method to insert a new row into the specified index of an unbound grid.

Complete the following steps:

1. Create a new .NET project.
2. Navigate to the Toolbox and add the **C1TrueDBGrid**, **Label**, **NumericUpDown**, and **Button** controls to the form.
3. Set the **Button1.Text** property to "Add New Row" and the **Label1.Text** property to "New Row Index".
4. Arrange the controls on the form similar to the following image:



- Switch to Code view and add the following imports (or using) statement to the project:

To write code in Visual Basic

Visual Basic

```
imports Cl.Win.ClTrueDBGrid
```

To write code in C#

C#

```
using Cl.Win.ClTrueDBGrid;
```

- Add the following code to create the **Form_Load** event and add data to the grid:

To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Add a caption to the grid.
    Me.ClTrueDBGrid1.Caption = "Unbound Grid"

    ' Add columns to the grid.
    Me.ClTrueDBGrid1.Columns.Add(New Cl.Win.ClTrueDBGrid.ClDataColumn("Col 1",
GetType(String)))
    Me.ClTrueDBGrid1.Columns.Add(New Cl.Win.ClTrueDBGrid.ClDataColumn("Col 2",
GetType(String)))
    Me.ClTrueDBGrid1.Columns.Add(New Cl.Win.ClTrueDBGrid.ClDataColumn("Col 3",
GetType(String)))
    Me.ClTrueDBGrid1.Columns.Add(New Cl.Win.ClTrueDBGrid.ClDataColumn("Col 4",
GetType(String)))

    ' Call the SetDataBinding method with no arguments.
    Me.ClTrueDBGrid1.SetDataBinding()
```

```

    ' Populate the grid.
    Dim i As Integer
    For i = 0 To 20 - 1
        Dim s As String = String.Format("Data {0};Data {1};Data {2}; Data {3}",
New Object() {i, i, i, i})
        Me.C1TrueDBGrid1.AddRow(s)
    Next i
End Sub

```

To write code in C#

C#

```

private void Form1_Load(object sender, EventArgs e)
{
    // Add a caption to the grid.
    this.c1TrueDBGrid1.Caption = "Unbound Grid"

    // Add columns to the grid.
    this.c1TrueDBGrid1.Columns.Add(new C1.Win.C1TrueDBGrid.C1DataColumn("Col 1",
typeof(string)));
    this.c1TrueDBGrid1.Columns.Add(new C1.Win.C1TrueDBGrid.C1DataColumn("Col 2",
typeof(string)));
    this.c1TrueDBGrid1.Columns.Add(new C1.Win.C1TrueDBGrid.C1DataColumn("Col 3",
typeof(string)));
    this.c1TrueDBGrid1.Columns.Add(new C1.Win.C1TrueDBGrid.C1DataColumn("Col 4",
typeof(string)));

    // Call the SetDataBinding method with no arguments.
    this.c1TrueDBGrid1.SetDataBinding();

    // Populate the grid.
    for (int i = 0; i < 20; i++)
    {
        string s = String.Format("Data {0};Data {1};Data {2}; Data {3}", i, i,
i, i);
        this.c1TrueDBGrid1.AddRow(s);
    }
}

```

7. Add the following code to create the **Button_Click** event and create a new row at the specified index when the button is clicked:

To write code in Visual Basic

Visual Basic

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim idx As Integer = CInt(Me.NumericUpDown1.Value)
    ' Create a new row.
    Dim dr As DataRow = Me.C1TrueDBGrid1.NewRow
    dr.Item(0) = "new row"
    ' Add the new row at the selected index.

```

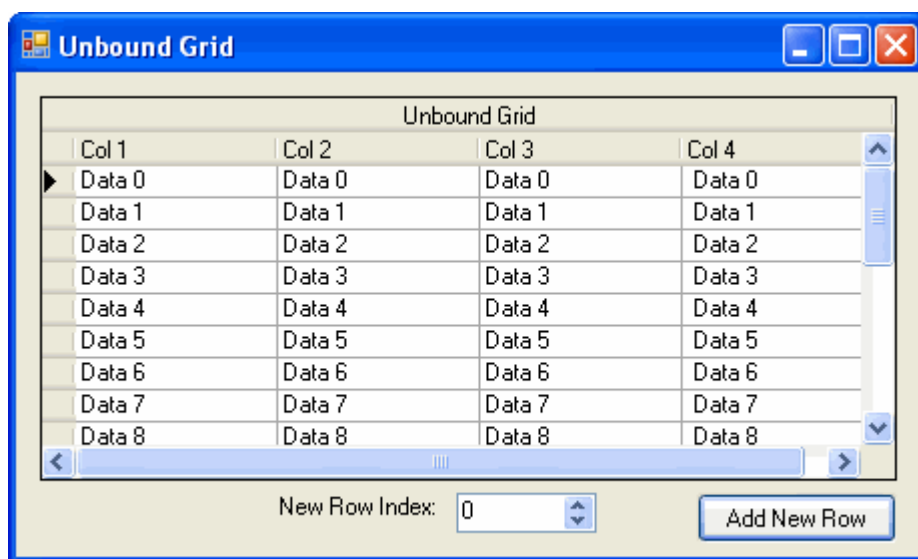
```
Me.clTrueDBGrid1.Rows.InsertAt(dr, idx)
End Sub
```

To write code in C#

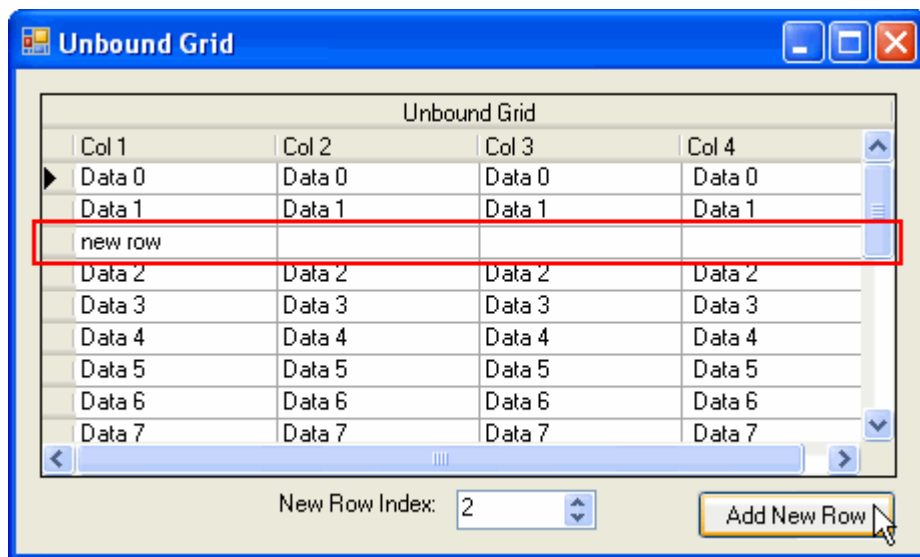
```
C#
private void button1_Click(object sender, EventArgs e)
{
    int idx = (int) this.numericUpDown1.Value;
    // Create a new row.
    DataRow dr = this.clTrueDBGrid1.NewRow();
    dr[0] = "new row";
    // Add the new row at the selected index.
    this.clTrueDBGrid1.Rows.InsertAt(dr, idx);
}
```

Run the application and observe:

The form will appear similar to the following:



Use the arrows to change the number in the **New Row Index** box and then select the **Add New Row** button. The new row will appear at the index that you chose:



Customizing the Grid's Appearance

The following topics explain how to configure the non-interactive elements of **True DBGrid for WinForms**' display, such as visual styles, captions, headers, footers, record selectors, and dividing lines.

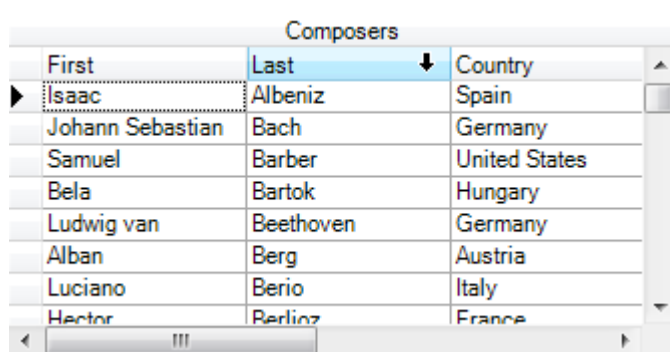
Visual Styles

True DBGrid for WinForms supports Visual Styles that mimic the styles available in Office 2007 and 2010.

Customizing Visual Styles simple, you can set the grid's [VisualStyle](#) Property from the **C1TrueDBGrid Tasks** menu (see [C1TrueDBGrid Tasks Menu](#) for more information), the Properties window, or in code. By default the grid's [VisualStyle](#) is set to [VisualStyle.Custom](#), a standard appearance that does not use Visual Styles and renders the control using only the set styles and properties. The following Visual Styles are available in **C1TrueDBGrid**:

- **Custom VisualStyle**

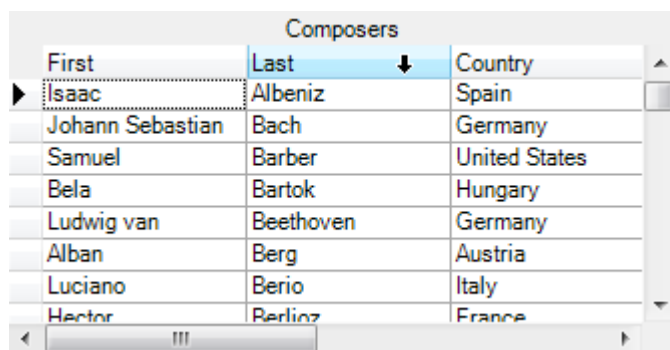
The Custom Visual Style renders the control using only the set styles and properties. This is the default setting. When VisualStyle is set to VisualStyle.Custom, the grid appears similar to the following:



| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

- **System VisualStyle**

The System Visual Style renders the control with an appearance based on the current system settings. When VisualStyle is set to VisualStyle.System, the grid appears similar to the following:



| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

- **Office2007Black VisualStyle**

The Office2007Black Visual Style renders the control with an appearance based on the Office 2007 Black color scheme. When VisualStyle is set to VisualStyle.Office2007Black, the grid appears similar to the following:

| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

- **Office2007Blue VisualStyle**

The Office2007Blue Visual Style renders the control with an appearance based on the Office 2007 Blue color scheme. When VisualStyle is set to VisualStyle.Office2007Blue, the grid appears similar to the following:

| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

- **Office2007Silver VisualStyle**

The Office2007Silver Visual Style renders the control with an appearance based on the Office 2007 Silver color scheme. When VisualStyle is set to VisualStyle.Office2007Silver, the grid appears similar to the following:

| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

- **Office2010Black VisualStyle**

The Office2010Black Visual Style renders the control with an appearance based on the Office 2010 Black color scheme. When VisualStyle is set to VisualStyle.Office2010Black, the grid appears similar to the following:

| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

- **Office2010Blue VisualStyle**

The Office2010Blue Visual Style renders the control with an appearance based on the Office 2010 Blue color scheme. When VisualStyle is set to VisualStyle.Office2010Blue, the grid appears similar to the following:

| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

- **Office2010Silver VisualStyle**

The Office2010Silver Visual Style renders the control with an appearance based on the Office 2010 Silver color scheme. When VisualStyle is set to VisualStyle.Office2010Silver, the grid appears similar to the following:

| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

Captions, Headers, and Footers

Affix a title to a grid, column, or split by setting the [Caption](#) property of the appropriate object. For example, the following code sets captions on a grid, column, and split:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Caption = "Grid Caption"
Me.C1TrueDBGrid1.Columns(0).Caption = "Column 0 Caption"
Me.C1TrueDBGrid1.Splits(0).Caption = "Split 0 Caption"
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Caption = "Grid Caption";  
this.c1TrueDBGrid1.Columns[0].Caption = "Column 0 Caption";  
this.c1TrueDBGrid1.Splits[0].Caption = "Split 0 Caption";
```

Column and Grid Captions

For [C1DataColumn](#) objects, the [Caption](#) property specifies the text that appears in each column's header area.

| Column 0 | Column 1 |
|----------|----------|
| * | |

If using **True DBGrid for WinForms** controls bound to a DataSet, the column captions are set automatically at run time.

Column captions can also be set in the designer using the **C1TrueDBGrid Designer**, or in code by manipulating the [C1DataColumnCollection](#).

The **Caption** property also applies to the [C1TrueDBGrid](#) control itself, which provides a descriptive header for the entire grid.

| TrueDBGrid | |
|------------|----------|
| Column 0 | Column 1 |
| * | |

By default, **C1TrueDBGrid** displays headings for each column; even the **Caption** property of an individual column is never set explicitly. However, all column headings can be hidden by setting the [ColumnHeaders](#) property to **False**.

| TrueDBGrid | |
|------------|--|
| * | |

Column Footers

Just as the [ColumnHeaders](#) property controls the display of column captions, the [ColumnFooters](#) property controls the display of the column footer row. Column footers, which are similar in appearance to column headers, are always displayed at the bottom of the grid, even if it is under populated.

| Column 0 | Column 1 |
|----------|----------|
| * | |
| Footer 0 | Footer 1 |

For each [C1DataColumn](#) object, the [FooterText](#) property determines the text that is displayed within the footer row.

Set the footer text in the designer using the **C1TrueDBGrid Designer**, or in code by manipulating the [C1DataColumnCollection](#) collection, as in the following example:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns(0).FooterText = "Footer 0"  
Me.C1TrueDBGrid1.Columns(1).FooterText = "Footer 1"
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Columns[0].FooterText = "Footer 0";  
this.c1TrueDBGrid1.Columns[1].FooterText = "Footer 1";
```

Unlike the [Caption](#) property, the **FooterText** property is not set automatically from a bound data source, so you will have to set it yourself.

Multiple-Line Headers and Footers

The split specific property [ColumnCaptionHeight](#) property controls the height of the column headers. By default, it is based upon the font setting of the HeadingStyle. To display more than one line of text in a column header, increase the **ColumnCaptionHeight** property to accommodate additional lines, as in the following example:

To write code in Visual Basic

Visual Basic

```
With Me.C1TrueDBGrid1  
    .Splits(0).ColumnCaptionHeight = .Splits(0).ColumnCaptionHeight * 2  
    .Columns(0).Caption = "First line" + vbCrLf + "Second line"  
End With
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].ColumnCaptionHeight =  
this.c1TrueDBGrid1.Splits[0].ColumnCaptionHeight * 2;  
this.c1TrueDBGrid1.Columns[0].Caption = "First line\nSecond line";
```

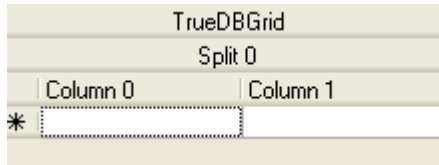
Note the use of the "\n" to specify a line break within the caption text. After this code executes, the first column's caption will contain two lines of text, and the second column's caption will be centered vertically.

| TrueDBGrid | |
|---------------------------|----------------------|
| First line Second line | Column 1 |
| * <input type="text"/> | <input type="text"/> |

Similarly, set the [ColumnFooterHeight](#) property to control the height of column footers, and use the constant to specify a line break when setting the [FooterText](#) property of a column.

Split Captions

[Split](#) objects can also have their own captions. For a grid with one split, a split caption can serve as a second grid caption.



However, split captions are best used in grids with at least two splits, as they are ideal for categorizing groups of columns for end users.

| Composers | | Vital Statistics | | |
|------------------|-----------|------------------|------------|------------|
| First | Last | Country | Birth | Death |
| Isaac | Albeniz | Spain | 5/29/1860 | 5/18/1909 |
| Johann Sebastian | Bach | Germany | | |
| Samuel | Barber | United States | 3/9/1910 | |
| Bela | Bartok | Hungary | 3/25/1881 | 9/26/1945 |
| Ludwig van | Beethoven | Germany | 12/16/1770 | 3/26/1827 |
| Alban | Berg | Austria | 2/9/1885 | 12/24/1935 |
| Luciano | Berio | Italy | 10/10/1925 | |
| Hector | Berlioz | France | 12/11/1803 | 3/8/1869 |
| Leonard | Bernstein | United States | 8/25/1918 | |
| Georges | Bizet | France | 10/25/1838 | 6/3/1875 |
| Ernest | Bloch | Switzerland | 7/24/1880 | 7/15/1959 |
| Alexander | Borodin | Russia | 11/12/1833 | 2/27/1887 |
| Johannes | Brahms | Germany | 5/7/1833 | 4/3/1897 |
| Benjamin | Britten | England | 11/22/1913 | 12/4/1976 |

Three-Dimensional vs. Flat Display

True DBGrid for WinForms supports a standard, "flat" control appearance, the more attractive three-dimensional appearance used by many controls, and a third that combines the flat appearance with the 3D. By default, the grid's [FlatStyle](#) property is set to [FlatModeEnum.Standard](#) so that the 3-D look is used. However, this property only controls whether 3D effects are used to draw the grid's border, caption bars, column headings and footings, and the record selector column. It does not affect the grid's data cells or row and column dividers. The following settings are available:

- When [FlatStyle](#) is set to [FlatModeEnum.Standard](#), the grid looks like this:

| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |

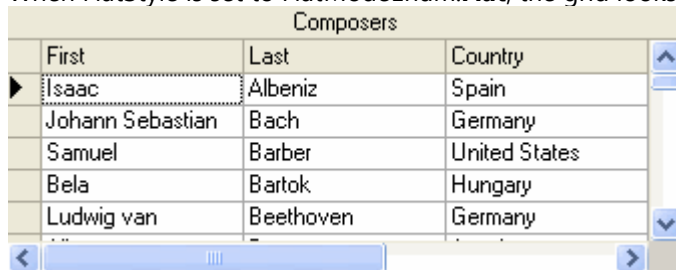
- When [FlatStyle](#) is set to [FlatModeEnum.PopUp](#), the grid looks like this:



| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |

Note that the initial grid has the same in appearance as FlatModeEnum.**Flat**. As the mouse moves over any control element, the appearance of that element takes on a 3D look.

- When FlatStyle is set to FlatModeEnum.**Flat**, the grid looks like this:



| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |

To achieve a 3D appearance for the entire grid, including its interior, set the following properties either in the designer or in code:

- In the Properties window, set the [RowDivider](#) style property to **Raised**. Or, in code:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.RowDivider.Style = C1.Win.C1TrueDBGrid.LineStyleEnum.Raised
```

To write code in C#

C#

```
this.c1TrueDBGrid1.RowDivider.Style = C1.Win.C1TrueDBGrid.LineStyleEnum.Raised;
```

- In the Splits Collection editor, set the [Style](#) property to [LineStyleEnum.Raised](#) for **all** [ColumnDivider](#) style objects for each split. Or, in code:

To write code in Visual Basic

Visual Basic

```
Dim C As C1.Win.C1TrueDBGrid.C1DisplayColumn
For Each C In Me.C1TrueDBGrid1.Splits(0).DisplayColumns
    C.ColumnDivider.Style = C1.Win.C1TrueDBGrid.LineStyleEnum.Inset
Next
```

To write code in C#

C#

```
C1.Win.C1trueDBGrid.C1DisplayColumn C ;
for each(C in this.C1trueDBGrid1.Splits[0].DisplayColumns)
{
```

```
C.ColumnDivider.Style = C1.Win.C1TrueDBGrid.LineStyleEnum.Raised;  
}
```

3. In the Properties window, set the **BackColor** property of the Normal style to **Lavender**. Or, in code:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Styles("Normal").BackColor = System.Drawing.Color.Lavender
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Styles["Normal"].BackColor = System.Drawing.Color.Lavender;
```

The resulting grid will look something like this:

| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |

Note that changing the **Style** property of the RowDivider object to **Raised** consumes an extra vertical pixel in each data row, resulting in fewer visible rows.

Try experimenting with other color combinations and divider styles to achieve different 3D effects, as explained in the [Borders and Dividing Lines](#) section.

Borders and Dividing Lines

The [RowDivider](#) and [ColumnDivider](#) properties enable different horizontal and vertical lines to be selected and also enable the color of the lines to be set. The properties return an underlying [GridLines](#) object that has two properties. These two properties, [Style](#) and [Color](#) define how the grid's cell borders will look. The allowable values for the **Style** property are as follows:

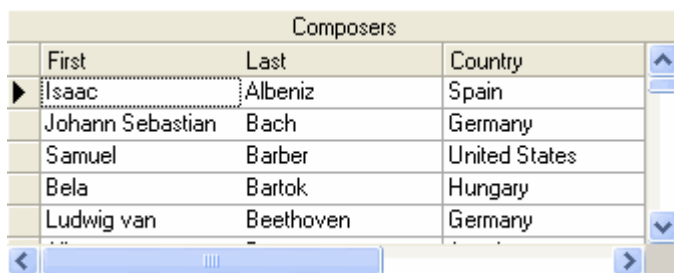
- [LineStyleEnum.Double](#)
- [LineStyleEnum.Inset](#)
- [LineStyleEnum.Raised](#)
- [LineStyleEnum.None](#)
- [LineStyleEnum.Single](#)

For example, setting the style property of RowDivider to [LineStyleEnum.None](#) eliminates the dividing lines between rows and enables you to cram a bit more data into the available area.



| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

For ColumnDivider properties, you can set the **Style** property to `LineStyleEnum.None`, and the `HeaderDivider` property to **False**. This enables you to visually group related columns, as shown in the following figure.



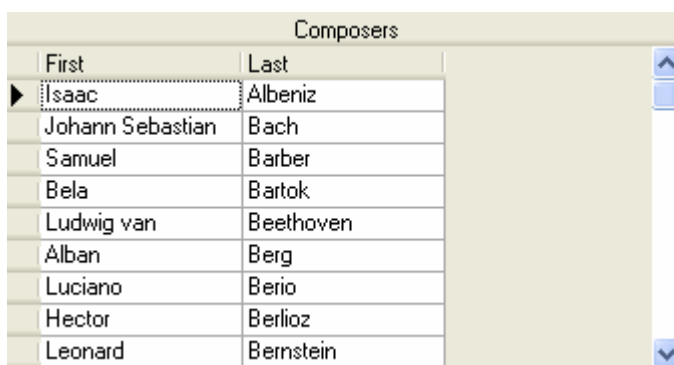
| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |

Unpopulated Regions

Depending upon the number of rows and columns in the data source, a portion of the grid's interior may not contain data cells. However, these "dead zones" can be eliminated using the `ExtendRightColumn` and `EmptyRows` properties. Change the color of the dead areas by using the `BackColor` property.

The Rightmost Column

As the grid scrolls horizontally until the last column is totally visible, there is usually a blank area between the last column and the right border of the grid.



| First | Last |
|------------------|-----------|
| Isaac | Albeniz |
| Johann Sebastian | Bach |
| Samuel | Barber |
| Bela | Bartok |
| Ludwig van | Beethoven |
| Alban | Berg |
| Luciano | Berio |
| Hector | Berlioz |
| Leonard | Bernstein |

The color of this blank area depends on the setting of your system's 3D Objects color (or Button Face color). Eliminate this blank area with the `ExtendRightColumn` property. The default value of this property is **False**, but if set it **True**, the last column will extend its width to the right edge of the grid.

| Composers | |
|------------------|-----------|
| First | Last |
| Isaac | Albeniz |
| Johann Sebastian | Bach |
| Samuel | Barber |
| Bela | Bartok |
| Ludwig van | Beethoven |
| Alban | Berg |
| Luciano | Berio |
| Hector | Berlioz |
| Leonard | Bernstein |

Unused Data Rows

If the data source contains fewer rows than the grid can display, the area below the AddNew row (or the last data row, if [AllowAddNew](#) is **False**) is left blank.

| American Composers | |
|--------------------|-----------|
| First | Last |
| Samuel | Barber |
| Leonard | Bernstein |
| Aaron | Copland |
| George | Gershwin |
| Charles | Ives |
| Virgil | Thomson |
| * | |

The color of this blank area depends on the setting of your system's 3D Objects color (or Button Face color). Eliminate this blank area with the [EmptyRows](#) property. The default value of this property is **False**, but if set to **True**, the grid will display empty rows below the last usable data row.

| American Composers | |
|--------------------|-----------|
| First | Last |
| Samuel | Barber |
| Leonard | Bernstein |
| Aaron | Copland |
| George | Gershwin |
| Charles | Ives |
| Virgil | Thomson |
| * | |
| | |
| | |
| | |
| | |
| | |

Note that the empty rows cannot receive focus.

Both the **EmptyRows** and [ExtendRightColumn](#) properties can be set to **True** to ensure that no blank areas appear

within the interior of the grid.

| American Composers | |
|--------------------|-----------|
| First | Last |
| ▶ Samuel | Barber |
| Leonard | Bernstein |
| Aaron | Copland |
| George | Gershwin |
| Charles | Ives |
| Virgil | Thomson |
| * | |
| | |
| | |
| | |
| | |
| | |

Highlighting the Current Row or Cell

The term *marquee* refers to the highlighted area that represents the current grid cell or row. The [MarqueeStyle](#) property can be set to several possible presentations, all enumerations of the [MarqueeEnum](#) object, illustrated as follows.

- **MarqueeEnum.DottedCellBorder**

The current cell is highlighted by a dotted border.

| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| ▶ Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

- **MarqueeEnum.SolidCellBorder**

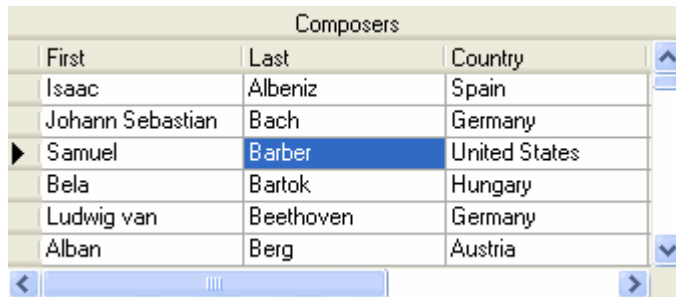
This is a more distinctive form of cell highlighting, often useful when a different background color is used (since the dotted rectangle is often difficult to spot).

| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| ▶ Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

- **MarqueeEnum.HighlightCell**

This style inverts the current cell completely, making it very visible. Values of the [BackColor](#) and [ForeColor](#)

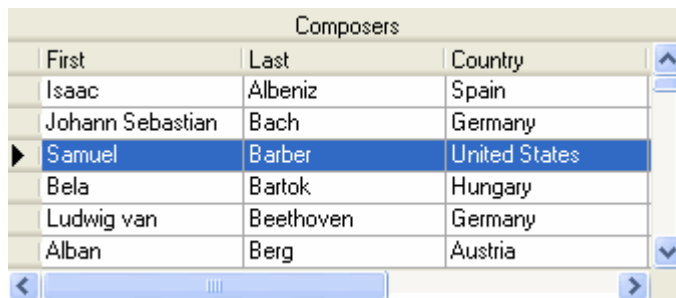
properties of the Edit Style should be chosen carefully to make a pleasing effect if the grid is editable.



| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

- **MarqueeEnum.HighlightRow**

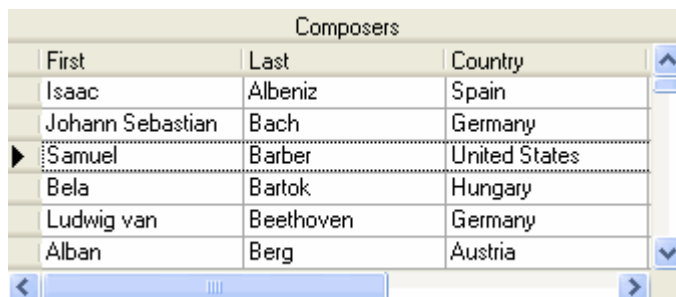
The entire row will be highlighted, but it will not be possible to tell which cell is the current cell in the row. To change highlight colors, edit the built-in **HighlightRow** style. See [Highlighting the Row of the Selected Cell](#) for more information. This style is most useful when the grid is not editable and users would view the data one record at a time.



| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

- **MarqueeStyleEnum.DottedRowBorder**

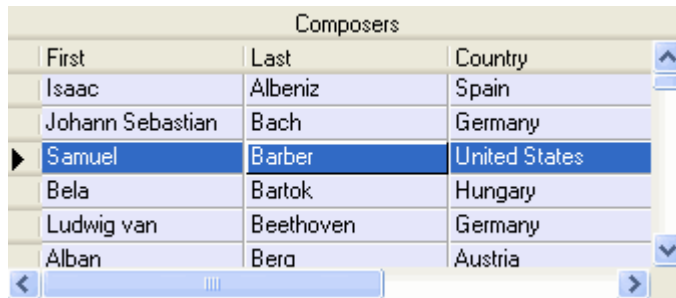
This setting highlights the entire row with a dotted rectangle. Use this setting instead of **HighlightRow** if a more subdued highlight is preferred.



| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

- **MarqueeEnum.HighlightRowRaiseCell**

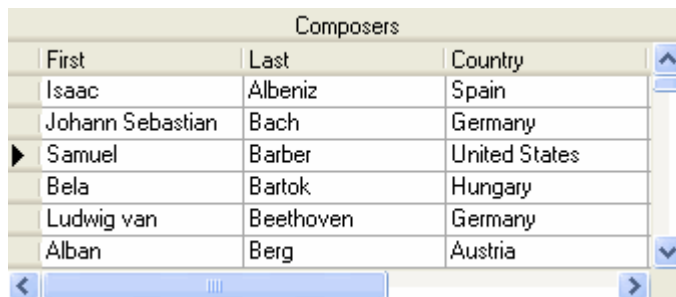
This value should only be used if 3D lines are used in the grid, since cell highlighting is accomplished using a "raised" appearance for the current cell.



| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| ▶ Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

- **MarqueeEnum.NoMarquee**

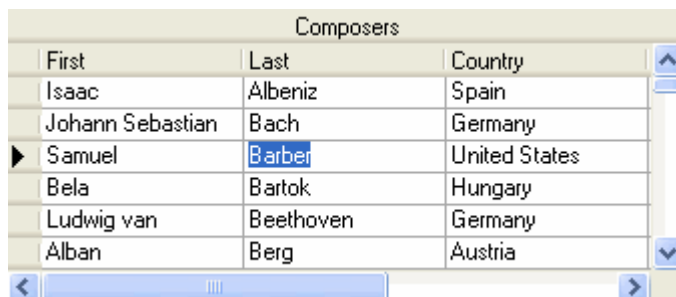
This setting will make the marquee disappear completely. Often this setting is useful for cases where the current row is irrelevant, or where you do not want to draw the user's attention to the grid until necessary.



| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| ▶ Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

- **MarqueeEnum.FloatingEditor**

This is the default marquee style of the grid. The cell text (the actual text only, **not** the entire cell) is highlighted and there is a blinking text cursor (caret) at the end of the text.



| Composers | | |
|------------------|-----------|---------------|
| First | Last | Country |
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| ▶ Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

The color of the highlight is your system's highlight color. The floating editor style simulates the look and feel of the Microsoft Access datasheet. The blinking text cursor indicates that the cell is *edit-ready*, hence the name floating editor for this marquee style. Since no other marquee style places the cell in a similar edit-ready mode, the behavior of the grid with the floating editor is sometimes different from the other marquee styles. The following list summarizes the differences when the [MarqueeStyle](#) property is set to [MarqueeEnum.FloatingEditor](#):

1. The following properties are ignored by the floating editor: [EditDropDown](#) and [EditorStyle](#).
2. When using the [AddCellStyle](#) and [AddRegexCellStyle](#) methods with the floating editor, the grid ignores the current cell bit ([CellStyleFlag.CurrentCell](#)) and highlighted row bit ([CellStyleFlag.MarqueeRow](#)) of the [Condition](#) parameter. For more details, see [Applying Styles to Cells](#).
3. The floating editor will not be displayed in a cell with radio buttons or a picture, as described in [Automatic Data Translation with ValueItems](#). A dotted cell marquee will be used instead. The floating editor highlight will return when the current cell is changed to one with normal text display.
4. The [CycleOnClick](#) property (applies to [ValueItemCollection](#)) has no effect when the [MarqueeStyle](#) property is set to [MarqueeEnum.FloatingEditor](#).
5. The **DoubleClick** event of the [C1TrueDBGrid](#) control does not fire when the user double-clicks a non-current cell within the grid. This is because the first click is used by the floating editor to begin editing, placing the cell

into edit mode at the character on which the click occurred. Double-clicking the current cell of the grid fires the **DoubleClick** event normally, however.

Row Height and Word Wrap

The following topics describe how to adjust the height of all rows in the grid using the [RowHeight](#) property and enabling word wrapping in cells using the [WrapText](#) property.

Adjusting the Height of All Grid Rows

Configure the row height interactively at design time by placing the grid in its visual editing mode or by changing the grid's [RowHeight](#) property in the Properties window. At run time, the user can adjust the row height interactively if [AllowRowSizing](#) is set to **RowSizingEnum.AllRows** or **RowSizingEnum.IndividualRows**. For more information, see [Run-Time Interaction](#).

The [RowHeight](#) property is expressed as pixels. However, a setting of 0 causes the grid to readjust its display so that each row occupies a single line of text in the current font. Therefore, use the following code to adjust the row height to display exactly three lines of text:

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.RowHeight = 0  
Me.ClTrueDBGrid1.RowHeight = 3 * Me.ClTrueDBGrid1.RowHeight
```

To write code in C#

C#

```
this.clTrueDBGrid1.RowHeight = 0;  
this.clTrueDBGrid1.RowHeight = 3 * this.clTrueDBGrid1.RowHeight;
```

This technique is particularly effective when displaying multiple-line memo fields, as in this example.

| Element | Description |
|-----------|---|
| Actinium | A radioactive chemical element found with uranium and radium in pitchblende and other minerals and formed in reactors by the neutron irradiation of radium. |
| Aluminum | A silvery, lightweight, easily worked metal that resists corrosion and is found abundantly, but only in combination. |
| Americium | One of the transuranic elements produced by the beta decay of an isotope of plutonium. |
| Antimony | A silvery-white, brittle, metallic chemical element of crystalline structure, found only in combination. It is used in alloys with other metals to harden them and increase |

Note that the Description column's Style object must have its [WrapText](#) property set to **True**; otherwise, the memo field display will be truncated after the first line.

Enabling Wordwrap in Cells

By default, a grid cell displays a single line of text, truncated at the cell's right boundary. Display multiple lines of text in a cell by increasing the grid's [RowHeight](#) property and setting the [WrapText](#) property of the desired column's [Style](#) object to **True**. If **WrapText** is **True** (the default is **False**), a line break occurs before words that would otherwise be partially displayed in a cell. The cell contents will continue to display on the next line, assuming that the grid's row height accommodates multiple lines.

Use the following loop to enable wordwrap for all grid columns:

To write code in Visual Basic

Visual Basic

```
Dim C As C1.Win.C1TrueDBGrid.C1DisplayColumn
For Each C In Me.C1TrueDBGrid1.Splits(0).DisplayColumns
    C.Style.WrapText = True
Next
```

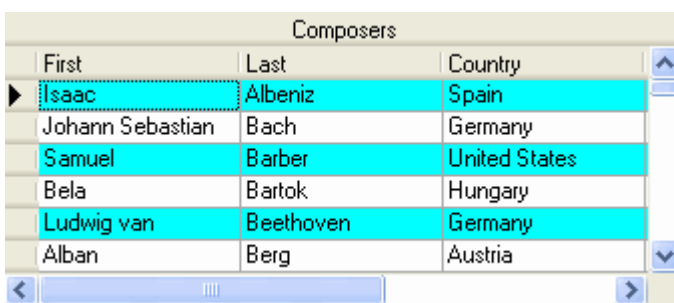
To write code in C#

C#

```
C1.Win.C1trueDBGrid.C1DisplayColumn C ;
for each(C in this.C1trueDBGrid1.Splits[0].DisplayColumns)
{
    C.Style.WrapText = true ;
}
```

Alternating Row Colors

The readability of the display can often be improved by alternating the background colors of adjacent rows. When the [AlternatingRows](#) property to **True** is set, the grid displays odd-numbered rows (the first displayed row is 1) using the built-in style **OddRow**, and even-numbered rows using the built-in style **EvenRow**.



The screenshot shows a TrueDBGrid control titled "Composers". It contains a table with three columns: "First", "Last", and "Country". The rows are: Isaac Albeniz (Spain), Johann Sebastian Bach (Germany), Samuel Barber (United States), Bela Bartok (Hungary), Ludwig van Beethoven (Germany), and Alban Berg (Austria). The rows are alternating between a light yellow background and a light blue background. The first row (Isaac Albeniz) is highlighted with a blue selection bar on the left.

| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

Horizontal and Vertical Alignment

Use the [HorizontalAlignment](#) property of the column's [Style](#) object to control the horizontal placement of cell text within a column. The allowable values for this property are as follows:

- [AlignHorzEnum.General](#)
- [AlignHorzEnum.Near](#)
- [AlignHorzEnum.Center](#)
- [AlignHorzEnum.Far](#)
- [AlignHorzEnum.Justify](#)

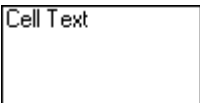
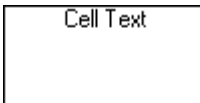
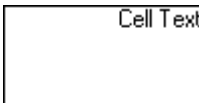
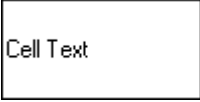
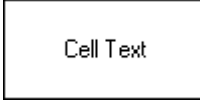
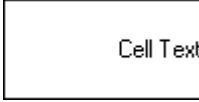
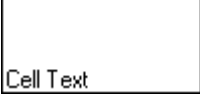
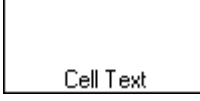
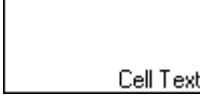
The setting [AlignHorzEnum.General](#), which is the default for data cells, indicates that the alignment should be based upon the underlying data type. For example, strings are left-aligned, while numbers are right-aligned.

Use the [VerticalAlignment](#) member of the [Style](#) object to control the vertical placement of text within the cells of a grid, split, or column. The allowable values for this property are as follows:

- [AlignVertEnum.Top](#)
- [AlignVertEnum.Center](#)
- [AlignVertEnum.Bottom](#)

For data cells, the default value is [AlignVertEnum.Top](#). For static grid elements such as caption bars, column headers, and column footers, the default value is [AlignVertEnum.Center](#). See the [Named Style Defaults](#) topic to learn how the default values are derived.

The following grid depicts all possible combinations of the [HorizontalAlignment](#) and [VerticalAlignment](#) properties.

| | AlignHorzEnum.Near | AlignHorzEnum.Center | AlignHorzEnum.Far |
|--------------------------------------|---|---|---|
| AlignVertEnum.Top |  |  |  |
| AlignVertEnum.Center |  |  |  |
| AlignVertEnum.Bottom |  |  |  |

The [AlignHorzEnum.General](#) and [AlignHorzEnum.Justify](#) settings have been omitted because the [AlignHorzEnum.General](#) setting aligns text as [AlignHorzEnum.Near](#) and numeric values as [AlignHorzEnum.Far](#). The [AlignHorzEnum.Justify](#) setting aligns text with respect to the cells boundaries, but in this case appears exactly like the [AlignHorzEnum.Near](#) setting.

The [HorizontalAlignment](#) and [VerticalAlignment](#) properties are tightly integrated with the concept of styles. For more information, see [How to Use Styles](#).

Data Presentation Techniques

This chapter explains how to display cell data in a variety of textual and graphical formats. To learn how to customize the behavior of cell editing in **True DBGrid for WinForms**, see [Cell Editing Techniques](#).

Text Formatting

In many cases, the raw numeric data that **True DBGrid for WinForms** receives from its data source is not suitable for end-user display. For example, date fields may need to be converted to a specific international format; currency fields may contain too many insignificant digits after the decimal point. Therefore, **True DBGrid for WinForms** provides a method with which you can alter the format of numerical fields, the [NumberFormat](#) property.

In addition, for situations where universal formatting of the database information is inadequate, **True DBGrid for WinForms** provides an event, [FormatText](#), that enables your application to override the default formatting on a per-column basis. By writing a simple handler for this event, you can customize the display of your column data in a myriad of ways.

Numeric Field Formatting

True DBGrid for WinForms supports a variety of data formatting options through the [C1DataColumn](#) object's [NumberFormat](#) property. The [NumberFormat](#) property reconfigures the data format that is handed to the grid from the database. It can alter most types of numeric values for a particular column.

For example, to display all date values within a column according to the form *26-Apr-01*, use the predefined *Medium Date* setting:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns("HireDate").NumberFormat = "Medium Date"
```

To write code in C#

C#

```
this.C1TrueDBGrid1.Columns["HireDate"].NumberFormat = "Medium Date";
```

Note that if the [NumberFormat](#) property of a column is changed at run time, the display does not need to refresh since **True DBGrid** handles this automatically.

Predefined Numeric Options

The [NumberFormat](#) property has several possible predefined options for numeric and time and date values.

For numeric data, the following predefined options are available in the [NumberFormat](#) property:

| Option | Description |
|-----------------------|---|
| <i>Standard</i> | Display number with thousands separator, at least one digit to the left and two digits to the right of the decimal separator. |
| <i>General Number</i> | Display number as is, with no thousand separators. |

| Option | Description |
|-------------------|---|
| <i>Currency</i> | Display number with thousands separator, if appropriate; display two digits to the right of the decimal separator. Note that output is based on system locale settings. |
| <i>Percent</i> | Display number multiplied by 100 with a percent sign (%) appended to the right; always display two digits to the right of the decimal separator. |
| <i>Fixed</i> | Display at least one digit to the left and two digits to the right of the decimal separator. |
| <i>Scientific</i> | Use standard scientific notation. |
| <i>Yes/No</i> | Display No if number is 0; otherwise, display Yes. |
| <i>True/False</i> | Display False if number is 0; otherwise, display True. |
| <i>On/Off</i> | Display Off if number is 0; otherwise, display On. |
| <i>0%</i> | Display number multiplied by 100, rounded to the nearest integer, with a percent sign (%) appended to the right. |
| <i>0.00%</i> | Same as <i>Percent</i> . |

For date and time data, the following predefined options are available in the [NumberFormat](#) property:

| Option | Description |
|---------------------|--|
| <i>General Date</i> | Display a date and/or time. For real numbers, display a date and time (for example, 4/3/93 05:34 PM); if there is no fractional part, display only a date (for example, 4/3/93); if there is no integer part, display only a time (for example, 05:34 PM). Date display is determined by your system settings. |
| <i>Long Date</i> | Display a date using your system's long date format. |
| <i>Medium Date</i> | Display a date using the medium date format appropriate for the language version of Visual Basic. |
| <i>Short Date</i> | Display a date using your system's short date format. |
| <i>Long Time</i> | Display a time using your system's long time format: includes hours, minutes, and seconds. |
| <i>Medium Time</i> | Display a time in 12-hour format using hours and minutes and the AM/PM designator. |
| <i>Short Time</i> | Display a time using the 24-hour format (for example, 17:45). |

Custom Number Formatting

You can customize the display of numeric information by setting the [NumberFormat](#) property to a custom value rather than to a predefined option.

For example to set a numeric column to specifically display with three decimal points, set the [NumberFormat](#) property using the following code:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns("Value").NumberFormat = "0.000"
```

To write code in C#

C#

```
this.C1TrueDBGrid1.Columns["Value"].NumberFormat = "0.000";
```

To set a date column to specifically display in the mm/dd/yyyy format, set the [NumberFormat](#) property using the following code:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns("BirthDate").NumberFormat = "MM/dd/yyyy"
```

To write code in C#

C#

```
this.C1TrueDBGrid1.Columns["BirthDate"].NumberFormat = "MM/dd/yyyy";
```

Input Validation with Built-In Formatting

It is important to note that the [NumberFormat](#) property affects only the *display* of data in the grid. Unless you also specify a value for the [EditMask](#) property, **True DBGrid for WinForms** does not enforce an input template, and the user is free to type anything into the formatted cell. When moving to another cell, the grid will reasonably interpret the user's input value and redisplay the data according to the [NumberFormat](#) setting.

For example, if *Medium Date* formatting is in effect for a column, a date of *Saturday, April 25, 1998, 12:00:00 AM* will be displayed as *25-Apr-98* with the day of the week and time ignored. If a user enters *July* and moves to another row, the grid cannot reasonably interpret the input date value and a trappable error will occur. If the user enters *oct 5* or *10/5*, the grid will interpret the entered date as *October 5, 2009* (that is, the current year is assumed). If the database update is successful, the entered date will be redisplayed as *05-Oct-09*, since *Medium Date* formatting is in effect.

Formatting with an Input Mask

Since it is common for the input and display formats to be the same, the [NumberFormat](#) property has an **Edit Mask** option (note the space between words). If this option is selected, then the [EditMask](#) property setting will be used for both data input and display. However, the input and display formats need not be the same, so you are free to select a [NumberFormat](#) option that differs from the [EditMask](#) property.

For example, the following code applies a phone number template to a column for both display and editing:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns("Phone").EditMask = "(###) ###-####"  
Me.C1TrueDBGrid1.Columns("Phone").NumberFormat = "Edit Mask"
```

To write code in C#

C#

```
this.clTrueDBGrid1.Columns["Phone"].EditMask = "(###) ###-####";  
this.clTrueDBGrid1.Columns["Phone"].NumberFormat = "Edit Mask";
```

For more information on how to specify a data input mask, see [Input Masking](#).

Formatting with a Custom Event Handler

On occasion, you may find that your current formatting options do not suit your particular needs. Furthermore, you may be restricted in the type of formatting that you can use or need a custom formatting option. In these cases, the **FormatText Event** option can be specified for the [NumberFormat](#) property. Choosing this option for a column will cause the [FormatText](#) event to fire each time data is about to be displayed in that column. The event allows you to reformat, translate, indent, or do anything you want to the data just prior to display:

To write code in Visual Basic

Visual Basic

```
Private Sub ClTrueDBGrid1_FormatText(ByVal sender As Object, ByVal e As  
Cl.Win.ClTrueDBGrid.FormatTextArgs) Handles ClTrueDBGrid1.FormatText  
  
End Sub
```

To write code in C#

C#

```
private void ClTrueDBGrid1_FormatText(object sender,  
Cl.Win.ClTrueDBGrid.FormatTextArgs e)  
{  
  
}
```

A member of the [FormatTextEventArgs](#) object, [ColIndex](#) is the column number of the grid to be reformatted. While the [Value](#) member contains the current value of the data and also serves as a placeholder for the formatted display value. For example, suppose the first column contains numeric values from 1 to 30, and you wish to display the data as Roman numerals:

To write code in Visual Basic

Visual Basic

```
Private Sub ClTrueDBGrid1_FormatText(ByVal sender As Object, ByVal e As  
Cl.Win.ClTrueDBGrid.FormatTextEventArgs) Handles ClTrueDBGrid1.FormatText  
  
    Dim result As String  
  
    If e.ColIndex = 0 Then  
  
        ' Determine how many X's.  
        While e.Value >= 10  
            result = result & "X"  
            e.Value = e.Value - 10  
        End While  
    End If  
  
    result = result & e.Value  
End Sub
```

```
End While

' Append "digits" 1-9.
Select Case e.Value
    Case 1
        result = result & "I"
    Case 2
        result = result & "II"
    Case 3
        result = result & "III"
    Case 4
        result = result & "IV"
    Case 5
        result = result & "V"
    Case 6
        result = result & "VI"
    Case 7
        result = result & "VII"
    Case 8
        result = result & "VIII"
    Case 9
        result = result & "IX"
End Select

' Change the actual format.
e.Value = result
End If
End Sub
```

To write code in C#

```
C#

private void ClTrueDBGrid1_FormatText(object sender,
Cl.Win.ClTrueDBGrid.FormatTextEventArgs e)

    string result;

    if ( e.ColIndex = 0 )
    {
        // Determine how many X's.
        while ( e.Value >= 10 )
        {
            result = result + "X";
            e.Value = e.Value - 10;
        }

        // Append "digits" 1-9.
        switch (e.Value)
        {
            case 1;
                result = result + "I";
```

```
        case 2;
            result = result + "II";
        case 3;
            result = result + "III";
        case 4;
            result = result + "IV";
        case 5;
            result = result + "V";
        case 6;
            result = result + "VI";
        case 7;
            result = result + "VII";
        case 8;
            result = result + "VIII";
        case 9;
            result = result + "IX";
    }

    // Change the actual format.
    e.Value = result;
}
}
```

Since the [FormatText](#) event has fewer restrictions than other formatting techniques, you can always use it to gain full control over the textual content of any value displayed in the grid.

Automatic Data Translation with Valueltems

Although the [FormatText](#) event can be used to map data values into more descriptive display values, **True DBGrid for WinForms** also provides a mechanism for performing such data translations automatically without code. Through the use of the [Valueltem](#) object, alternate text or even pictures can be specified to be displayed in place of the underlying data values.

This feature is ideally suited for displaying numeric codes or cryptic abbreviations in a form that makes sense to end-users. For example, country codes can be rendered as proper names or even as pictures of their respective flags. Or, the numbers 0, 1, and 2 may be displayed as Yes, No, and Maybe. Either the actual values (0, 1, 2) or the translated values (Yes, No, Maybe) may be displayed as radio buttons in a cell or in a drop-down combo box.

What are Valueltems?

The [Valueltems](#) object contains a collection and properties that define the association between an underlying data value and its visual representation within the grid. The [Valueltems](#) object contains a [ValueltemCollection](#) of zero or more [Valueltem](#) objects. Each [Valueltem](#) supports two main properties: [Value](#), the underlying data value, and [DisplayValue](#), its visual representation. Both properties are of type [Object](#). Additionally, each [C1DataColumn](#) object contains [Valueltems](#) object.

In code, manipulate the collection of [Valueltem](#) pairs as you would any other **True DBGrid for WinForms** or Visual Studio collection. [Valueltems](#) can be added, removed, or manipulated through the [ValueltemCollection](#) object.

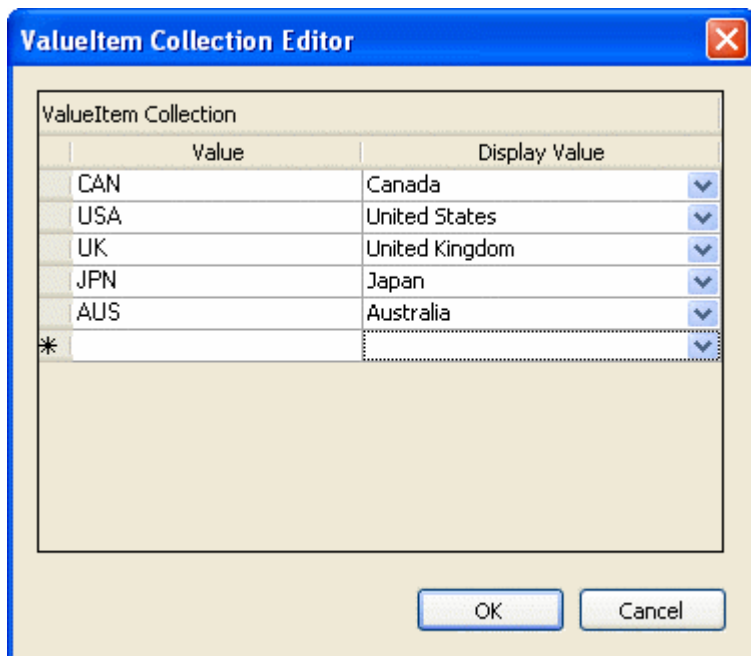
At design time a **Valueltem Collection Editor** is available through the **C1TrueDBGrid Designer**. For more information see [Using the ValueltemCollection Editor](#).

Specifying Text-to-Text Translations

Consider the following example, in which the *Country* field is represented by a short character code.

| Company | Country |
|-----------------------------------|---------|
| Maple Leaf Systems | CAN |
| Her Majesty's Software | UK |
| Software Mart | USA |
| Far East Distributors | JPN |
| Outback Software, Inc. | AUS |
| Northwest Purchasing Agents, Inc. | USA |

To display the character codes as proper names, use the column's [ValueItemCollection](#) object to specify automatic data translations. At design time, this is done with .NET's [ValueItemCollection](#) editor.



Altering the [ValueItemCollection](#) object through the collection editor enables you to specify data translations on a per-column basis in a simple window. To construct a list of data translations for an individual column, complete the following steps:

1. Open up the **C1TrueDBGrid Designer** by clicking on the **ellipsis** button (...) next to the **Columns collection** in the Properties window.
2. Select the column whose contents you would like translated. In the left pane expand the **ValueItems** node. Clicking on the **ellipsis** button next to the **Values** node will bring up the [ValueItemCollection](#) editor.
3. In the right pane under the **ValueItems** node, set the [Translate](#) property to **True**.
4. Clicking on the **Add** button in the left pane will add [ValueItem](#) objects. In the right pane specify a [Value](#) and [DisplayValue](#) for each [ValueItem](#). When entering the [ValueItem](#) text, disregard the **drop-down** button. This is used for entering a bitmap as a [DisplayValue](#).
5. Select **OK** or **Apply** to commit the changes.

When the program is run, *Country* field values that match any items in the [Value](#) column appear as the corresponding [DisplayValue](#) entry. For example, *CAN* becomes *Canada*, *UK* becomes *UnitedKingdom*, and so on.

| Company | Country |
|-----------------------------------|----------------|
| Maple Leaf Systems | Canada |
| Her Majesty's Software | United Kingdom |
| Software Mart | United States |
| Far East Distributors | Japan |
| Outback Software, Inc. | Australia |
| Northwest Purchasing Agents, Inc. | United States |

Note that the underlying database is not affected; only the presentation of the data value is different. The same effect can be achieved in code as follows:

To write code in Visual Basic

Visual Basic

```
Dim v as Cl.Win.ClTrueDBGrid.ValueItemCollection
v = Me.ClTrueDBGrid1.Columns("Country").ValueItems.Values

v.Add(new Cl.Win.ClTrueDBGrid.ValueItem("CAN", "Canada"))
v.Add(new Cl.Win.ClTrueDBGrid.ValueItem("UK", "United Kingdom"))
v.Add(new Cl.Win.ClTrueDBGrid.ValueItem("USA", "United States"))
v.Add(new Cl.Win.ClTrueDBGrid.ValueItem("JPN", "Japan"))
v.Add(new Cl.Win.ClTrueDBGrid.ValueItem("AUS", "Australia"))

Me.ClTrueDBGrid1.Columns("Country").ValueItems.Translate = True
```

To write code in C#

C#

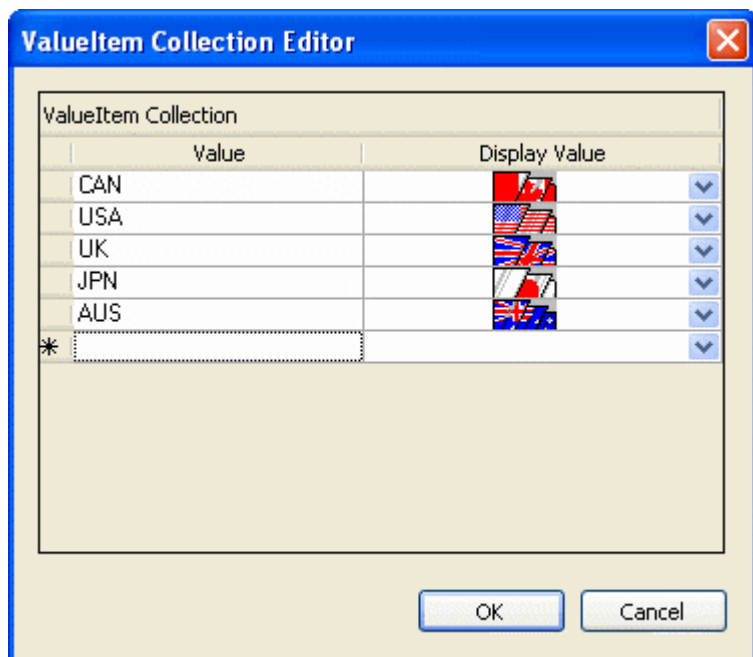
```
Cl.Win.ClTrueDBGrid.ValueItemCollection v =
this.ClTrueDBGrid1.Columns["Country"].ValueItems.Values;

v.Add(new Cl.Win.ClTrueDBGrid.ValueItem("CAN", "Canada"));
v.Add(new Cl.Win.ClTrueDBGrid.ValueItem("UK", "United Kingdom"));
v.Add(new Cl.Win.ClTrueDBGrid.ValueItem("USA", "United States"));
v.Add(new Cl.Win.ClTrueDBGrid.ValueItem("JPN", "Japan"));
v.Add(new Cl.Win.ClTrueDBGrid.ValueItem("AUS", "Australia"));

this.ClTrueDBGrid1.Columns["Country"].ValueItems.Translate = true;
```







Specifying Text-to-Picture Translations

The same techniques used to specify text-to-text translations can also be used for text-to-picture translations. Within the **ValueItem Collection Editor**, instead of typing a string into the [DisplayValue](#) column, use the **ellipsis** button (...) to select a bitmap to be used for data translations. To delete your bitmap selection, simply delete the text in the [DisplayValue](#) property box and either select another bitmap or type in text.



Note that the [Translate](#) property for the [ValueItems](#) object must be set to **True**. Depending upon the height of the bitmaps, it may be necessary to increase the value of the [RowHeight](#) property. If that is done, change the [VerticalAlignment](#) member of the grid's [Style](#) property to **Center** to ensure that the bitmaps (as well as textual data in other columns) are centered vertically within grid cells instead of being anchored at the top.

When the program is run, *Country* field values that match an item in the [Value](#) column appear as the corresponding [DisplayValue](#) picture:

| Company | Country |
|-----------------------------------|---|
| Maple Leaf Systems |  |
| Her Majesty's Software |  |
| Software Mart |  |
| Far East Distributors |  |
| Outback Software, Inc. |  |
| Northwest Purchasing Agents, Inc. |  |

As with textual translations, the underlying database is not affected; only the presentation of the data value is different. The same effect can be achieved in code as follows:

To write code in Visual Basic

Visual Basic

```
Dim item As C1.Win.C1TrueDBGrid.ValueItem = New C1.Win.C1TrueDBGrid.ValueItem()
With Me.C1TrueDBGrid1.Columns("Country").ValueItems.Values
    Item.Value = "CAN"
    Item.DisplayValue = System.Drawing.Image.FromFile("canada.bmp")
    .Add(Item)

    Item = New C1.Win.C1TrueDBGrid.ValueItem()
    Item.Value = "UK"
    Item.DisplayValue = System.Drawing.Image.FromFile("uk.bmp")
    .Add(Item)
```



```
Item = New Cl.Win.ClTrueDBGrid.ValueItem()  
Item.Value = "USA"  
Item.DisplayValue = System.Drawing.Image.FromFile("usa.bmp")  
.Add(Item)  
  
Item = New Cl.Win.ClTrueDBGrid.ValueItem()  
Item.Value = "JPN"  
Item.DisplayValue = System.Drawing.Image.FromFile("japan.bmp")  
.Add(Item)  
  
Item = New Cl.Win.ClTrueDBGrid.ValueItem()  
Item.Value = "AUS"  
Item.DisplayValue = System.Drawing.Image.FromFile("australia.bmp")  
.Add(Item)  
  
Me.ClTrueDBGrid1.Columns("Country").ValueItems.Translate = True  
End With
```

To write code in C#

C#

```
Cl.Win.ClTrueDBGrid.ValueItemCollection v =  
this.clTrueDBGrid.Columns["Country"].ValueItems.Values;  
Cl.Win.ClTrueDBGrid.ValueItem Item = new Cl.Win.ClTrueDBGrid.ValueItem();  
  
Item.value = "CAN";  
Item.DisplayValue = System.Drawing.Image.FromFile["canada.bmp"];  
v.Add[Item];  
  
Item = new Cl.Win.ClTrueDBGrid.ValueItem();  
Item.value = "UK";  
Item.DisplayValue = System.Drawing.Image.FromFile["uk.bmp"];  
v.Add[Item];  
  
Item = new Cl.Win.ClTrueDBGrid.ValueItem();  
Item.value = "USA";  
Item.DisplayValue = System.Drawing.Image.FromFile["usa.bmp"];  
v.Add[Item];  
  
Item = new Cl.Win.ClTrueDBGrid.ValueItem();  
Item.value = "JPN";  
Item.DisplayValue = System.Drawing.Image.FromFile["japan.bmp"];  
v.Add[Item];  
  
Item = new Cl.Win.ClTrueDBGrid.ValueItem();  
Item.value = "AUS";  
Item.DisplayValue = System.Drawing.Image.FromFile["australia.bmp"];  
v.Add[Item];  
  
this.clTrueDBGrid1.Columns["Country"].ValueItems.Translate = true;
```

Displaying Both Text and Pictures in a Cell

Once the [ValueItems](#) object is configured to perform text-to-picture translations for a column, you can display both the [Value](#) string and the [DisplayValue](#) bitmap to appear within the same cell by selecting the [AnnotatePicture](#) property. Or, in code:

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.Columns("Country").ValueItems.AnnotatePicture = True
```

To write code in C#

C#

```
this.ClTrueDBGrid1.Columns["Country"].ValueItems.AnnotatePicture = true;
```

The horizontal placement of the bitmap with respect to the cell text is determined by the [HorizontalAlignment](#) and [ForegroundPicturePosition](#) properties of the column's [Style](#) object. The enumeration objects for these two properties are the [AlignHorzEnum](#) and [ForegroundPicturePositionEnum](#) objects respectively. In the following example, [HorizontalAlignment](#) is set to [AlignHorzEnum.General](#). Since the *Country* column represents a string field, the cell text is left-aligned. However, since the [ForegroundPicturePosition](#) property is set to the default value of [ForegroundPicturePositionEnum.Near](#), the bitmap is placed at the left edge of the cell, and the cell text is left-aligned in the remaining space.


| Company | Country |
|-----------------------------------|---|
| Maple Leaf Systems |  CAN |
| Her Majesty's Software |  UK |
| Software Mart |  USA |
| Far East Distributors |  JPN |
| Outback Software, Inc. |  AUS |
| Northwest Purchasing Agents, Inc. |  USA |

However, if you change the [ForegroundPicturePosition](#) property to [ForegroundPicturePositionEnum.Far](#), then the cell text is left-aligned as usual, but the bitmap is right-aligned.

| Company | Country |
|-----------------------------------|---|
| Maple Leaf Systems | CAN  |
| Her Majesty's Software | UK  |
| Software Mart | USA  |
| Far East Distributors | JPN  |
| Outback Software, Inc. | AUS  |
| Northwest Purchasing Agents, Inc. | USA  |

To place the cell text below the bitmap while centering both items, set the [HorizontalAlignment](#) property to [AlignHorzEnum.Center](#) and the [ForegroundPicturePosition](#) property to [ForegroundPicturePositionEnum.TopofText](#).

| Company | Country |
|-----------------------------------|--|
| Maple Leaf Systems |  CAN |
| Her Majesty's Software |  UK |
| Software Mart |  USA |
| Far East Distributors |  JPN |
| Outback Software, Inc. |  AUS |
| Northwest Purchasing Agents, Inc. |  USA |

 **Note:** For an illustration of all possible combinations of the [HorizontalAlignment](#) and [ForeColor](#) properties, see [Displaying Foreground Pictures](#).

When editing, the editor uses all space available in the text portion of the cell. When the [Presentation](#) property of the [ValueItemCollection](#) object is set to one of the combo box options, the bitmap will not change until editing is completed.

Note that in the preceding examples, the text is displayed as it is stored in the database without formatting. Since the [ValueItem](#) object can only accommodate one translation, displaying both a picture and formatted text cannot be accomplished with [ValueItems](#) alone. Therefore, use the [FormatText](#) event to translate the text, and then use the [ValueItems](#) to associate the **translated text** (not the underlying data value) with a picture:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns("Country").NumberFormat = "FormatText Event"
```

To write code in C#

C#

```
this.C1TrueDBGrid1.Columns["Country"].NumberFormat = "FormatText Event";
```

In this example, the [NumberFormat](#) property is set to a special value that causes the [FormatText](#) event to fire:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_FormatText(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.FormatTextEventArgs) Handles C1TrueDBGrid1.FormatText

    Select Case e.Value
        Case "CAN"
            e.Value = "Canada"
        Case "UK"
            e.Value = "United Kingdom"
        Case "USA"
            e.Value = "United States"
        Case "JPN"
```

```
        e.Value = "Japan"
    Case "AUS"
        e.Value = "Australia"
    End Select
End Sub
```







To write code in C#

C#

```
private void ClTrueDBGrid1_FormatText( object sender,
Cl.Win.ClTrueDBGrid.FormatTextEventArgs e)
{
    switch (e.value)
    {
        case "CAN":
            e.value = "Canada";
            break;
        case "UK":
            e.value = "United Kingdom";
            break;
        case "USA":
            e.value = "United States";
            break;
        case "JPN":
            e.value = "Japan";
            break;
        case "AUS":
            e.value = "Australia";
            break;
    }
}
```

Since the [FormatText](#) event now translates the country codes stored in the database into actual names for display, the [Value](#) property of each [ValueItem](#) in the [ValueItemCollection](#) object must be changed accordingly.

Assuming that the [HorizontalAlignment](#) and [ForegroundPicturePosition](#) properties are set as in the previous example, the end result is that the underlying data is displayed as both descriptive text *and* a picture.

| Company | Country |
|-----------------------------------|---|
| Maple Leaf Systems |  Canada |
| Her Majesty's Software |  United Kingdom |
| Software Mart |  United States |
| Far East Distributors |  Japan |
| Outback Software, Inc. |  Australia |
| Northwest Purchasing Agents, Inc. |  United States |

Note: [DisplayValue](#) pictures are ideally suited for translating status codes or other fields where the number of allowable values is relatively small. To get a more generalized picture display mechanism than the [ValueItemCollection](#) object offers, use the [ForegroundPicturePosition](#) property in conjunction with the [FetchCellStyle](#) event to display arbitrary pictures on a per-cell basis. For more information, see [Displaying Foreground Pictures](#).

Displaying Boolean Values as Check Boxes

Use the [ValueItems](#) object to represent Boolean values as in-cell checkboxes. The effect of a working Boolean checkbox can be achieved without defining any [ValueItem](#) objects—just set the [Presentation](#) property to [PresentationEnum.CheckBox](#).

Note that the [Translate](#) checkbox does not need to be selected to enable automatic data translation, nor does the [CycleOnClick](#) checkbox need to be selected to enable end users to toggle the value of a cell by clicking it. The following figure shows a typical checkbox display.

| ProductName | QuantityPerUnit | Discontinued |
|--------------------|---------------------|-------------------------------------|
| Chai | 10 boxes x 20 bag | <input type="checkbox"/> |
| Chang | 24 - 12 oz bottles | <input checked="" type="checkbox"/> |
| Aniseed Syrup | 12 - 550 ml bottles | <input type="checkbox"/> |
| Chef Anton's Cajun | 48 - 6 oz jars | <input type="checkbox"/> |
| Chef Anton's Gum | 36 boxes | <input checked="" type="checkbox"/> |
| Grandma's Boysen | 12 - 8 oz jars | <input type="checkbox"/> |
| Uncle Bob's Organ | 12 - 1 lb pkgs. | <input type="checkbox"/> |
| Northwoods Cranb | 12 - 12 oz jars | <input checked="" type="checkbox"/> |

Note: To use different check box bitmaps, define a two-state collection of [ValueItem](#) objects through the [Values](#) property of the [C1DataColumn](#). Set the [Presentation](#) property to [PresentationEnum.Normal](#), and set the [Translate](#) and [CycleOnClick](#) properties to **True**.

Displaying Allowable Values as Radio Buttons

If the number of allowable values for a column is relatively small, consider a radio button presentation. At design time, go to the **C1TrueDBGrid Designer**, then to the [ValueItems](#) node for the column and set the [Presentation](#) property to

[PresentationEnum.RadioButton](#). Or, in code:

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.Columns("Country").ValueItems.Presentation =  
PresentationEnum.RadioButton
```

To write code in C#

C#

```
this.ClTrueDBGrid1.Columns["Country"].ValueItems.Presentation =  
PresentationEnum.RadioButton;
```

If necessary, adjust the [Width](#) property of the column style and the [RowHeight](#) property of the grid to accommodate all of the items.

| Company | Country |
|------------------------|--|
| Maple Leaf Systems | <input type="radio"/> United States <input type="radio"/> Japan <input checked="" type="radio"/> Canada <input type="radio"/> Australia <input type="radio"/> United Kingdom <input type="radio"/> Other |
| Her Majesty's Software | <input type="radio"/> United States <input type="radio"/> Japan <input type="radio"/> Canada <input type="radio"/> Australia <input checked="" type="radio"/> United Kingdom <input type="radio"/> Other |
| Software Mart | <input checked="" type="radio"/> United States <input type="radio"/> Japan <input type="radio"/> Canada <input type="radio"/> Australia <input type="radio"/> United Kingdom <input type="radio"/> Other |
| Far East Distributors | <input type="radio"/> United States <input checked="" type="radio"/> Japan <input type="radio"/> Canada <input type="radio"/> Australia <input type="radio"/> United Kingdom <input type="radio"/> Other |

For a given cell, if the underlying data does not match any of the available values, none of the radio buttons will be selected for that cell. However, a default [ValueItem](#) object can be provided that will be selected instead.

In this example, the last [ValueItem](#) has an empty [Value](#) property, but the [DisplayValue](#) is set to **Other**. This means that when *Country* fields that do not match any of the items are displayed their value in the grid will be displayed as **Other**.

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.Columns("Country").ValueItems.DefaultItem = 5
```

To write code in C#

C#

```
this.ClTrueDBGrid1.Columns["Country"].ValueItems.DefaultItem = 5;
```

In this example, 5 is the zero-based index of the default item.

Context-Sensitive Help with CellTips

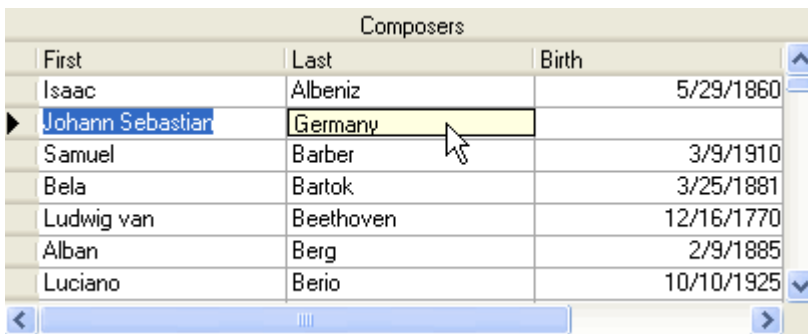
In many Windows applications, when the user points to a toolbar button and leaves the mouse at rest for a short time,

a ToolTip window appears with the name of the associated command. Provide similar context-sensitive help for users with the [CellTips](#) property of **True DBGrid for WinForms**.

The [CellTips](#) property determines whether the grid displays a pop-up text window when the cursor is idle. By default, this property is set to [CellTipEnum.NoCellTips](#), and cell tips are not displayed.

If the [CellTips](#) property is set to either [CellTipEnum.Anchored](#) or [CellTipEnum.Floating](#), the [FetchCellTips](#) event will be fired whenever the grid has focus and the cursor is idle over a grid cell, record selector, column header, column footer, split header, or grid caption. The event will not fire if the cursor is over the scroll bars.

The setting [CellTipEnum.Anchored](#) aligns the cell tip window with either the left or right edge of the cell. The left edge is favored, but the right edge will be used if necessary in order to display as much text as possible.



| Composers | | |
|------------------|-----------|------------|
| First | Last | Birth |
| Isaac | Albeniz | 5/29/1860 |
| Johann Sebastian | Germany | |
| Samuel | Barber | 3/9/1910 |
| Bela | Bartok | 3/25/1881 |
| Ludwig van | Beethoven | 12/16/1770 |
| Alban | Berg | 2/9/1885 |
| Luciano | Berio | 10/10/1925 |

The setting [CellTipEnum.Floating](#) displays the cell tip window below the cursor, if possible.



| Composers | | |
|------------------|-----------|------------|
| First | Last | Birth |
| Isaac | Albeniz | 5/29/1860 |
| Johann Sebastian | Bach | |
| Samuel | Barber | 3/9/1910 |
| Bela | Bartok | Germany |
| Ludwig van | Beethoven | 12/16/1770 |
| Alban | Berg | 2/9/1885 |
| Luciano | Berio | 10/10/1925 |

If a handler is not provided for the [FetchCellTips](#) event, and the cursor is over a grid cell, the default behavior is to display a text box containing the cell's contents (up to 256 characters). This enables the user to peruse the contents of a cell even if it is not big enough to be displayed in its entirety. The [FetchCellTips](#) event can be programmed to override the default cell text display in order to provide users with context-sensitive help.

A common application of the [FetchCellTips](#) event is to display the contents of an invisible column that provides additional information about the row being pointed to, as in the following example:

To write code in Visual Basic

Visual Basic

```
' General Declarations.
Dim DescCol As C1.Win.C1TrueDBGrid.C1DataColumn

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles MyBase.Load
        ' Set the column to be displayed as a CellTip.
        DescCol = Me.C1TrueDBGrid1.Columns("Country")
        Me.C1TrueDBGrid1.CellTips = C1.Win.C1TrueDBGrid.CellTipEnum.Floating
```

```
End Sub

Private Sub C1TrueDBGrid1_FetchCellTips(ByVal sender As System.Object, ByVal e As
C1.Win.C1TrueDBGrid.FetchCellTipsEventArgs) Handles C1TrueDBGrid1.FetchCellTips
    ' Display the column.
    e.CellTip = DescCol.CellText(e.Row)
End Sub
```

To write code in C#

```
C#

// General Declarations.
C1.Win.C1TrueDBGrid.C1DataColumn DescCol;

private void Form1_Load(System.Object sender, System.EventArgs e)
{
    // Set the column to be displayed as a CellTip.
    DescCol = this.c1TrueDBGrid1.Columns["Country"];
    this.c1TrueDBGrid1.CellTips = C1.Win.C1TrueDBGrid.CellTipEnum.Floating;
}

private void C1TrueDBGrid1_FetchCellTips(System.Object sender,
C1.Win.C1TrueDBGrid.FetchCellTipsEventArgs e)
{
    // Display the column.
    e.CellTip = DescCol.CellText(e.Row);
}
```

Use the [CellTipsDelay](#) property to control the amount of time that must elapse before the cell tip window is displayed.

Use the [CellTipsWidth](#) property to control the width of the cell tip window.

Scroll Tracking and ScrollTips

If the [ScrollTrack](#) property is set to **True**, moving the scrollbar thumb causes vertical scrolling of the grid's display. By default, this property is **False**, and no scrolling occurs until the thumb is released.

If the [ScrollTips](#) property is set to **True**, moving the scrollbar thumb causes the [FetchScrollTips](#) event to fire. Use this event to track the position of the scroll bar on a record-by-record basis. Also use this event to present the user with useful information relating to the current record or recordset. When used in tandem, the [ScrollTrack](#) and [ScrollTips](#) properties provide users with visual feedback when scrolling through large DataSets.



See [Tutorial 22: Borders, Scroll Tracking, and Scroll Tips](#) for more information.

Data-Sensitive Cell Merging

If the underlying grid data is sorted, the readability of the display may be improved by grouping adjacent like-valued cells within the sorted column(s). The `Merge` property of the `C1DisplayColumn` object controls whether its data cells are grouped in this manner to form a single non-editable cell, using the `ColumnMergeEnum`. By default, this property is set to **None**, and each physical row within a column displays a data value, if any.

Consider the following grid, which is sorted by the *Country* field.

| Country | First | Last |
|----------------|-----------------|-------------|
| Austria | Alban | Berg |
| Austria | Anton | Bruckner |
| Austria | Gustav | Mahler |
| Austria | Wolfgang Amadeu | Mozart |
| Austria | Arnold | Schoenberg |
| Austria | Franz | Schubert |
| Austria | Anton von | Webern |
| Brazil | Heitor | Villa-Lobos |
| Czechoslovakia | Antonin | Dvorak |
| Czechoslovakia | Leos | Janacek |
| Czechoslovakia | Bohuslav | Martinu |
| Czechoslovakia | Bedrich | Smetana |
| Denmark | Carl | Nielsen |
| England | Benjamin | Britten |
| Finland | Edvard | Grieg |

If data-sensitive cell merging is enabled for the *Country* column at run time, then its cells are grouped according to their contents. For example:

To write code in Visual Basic

Visual Basic

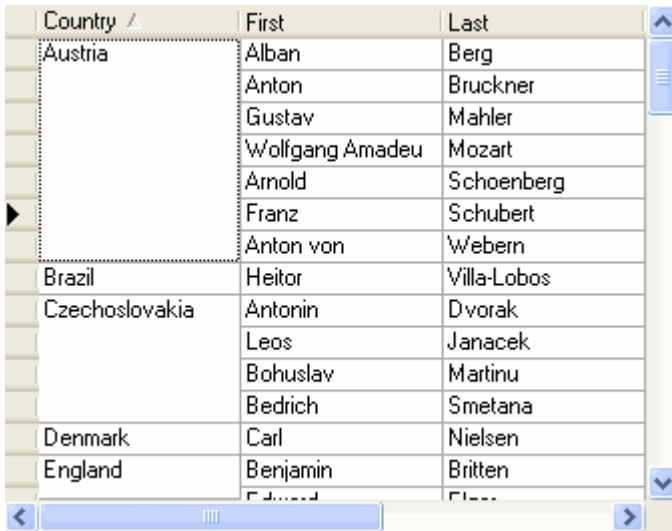
```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns("Country").Merge =  
C1.Win.C1TrueDBGrid.ColumnMergeEnum.Free
```

To write code in C#

C#

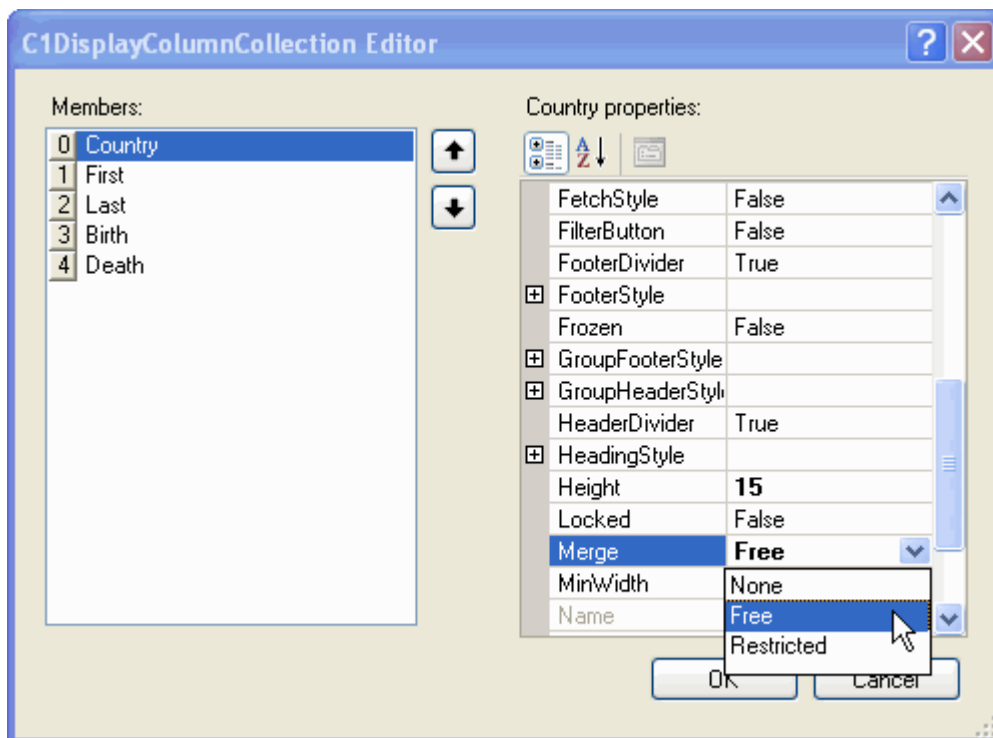
```
this.c1TrueDBGrid1.Splits[0].DisplayColumns["Country"].Merge =  
C1.Win.C1TrueDBGrid.ColumnMergeEnum.Free;
```

Executing this statement produces the following display. Note that when the current cell is in the *Country* column, the marquee spans all like-valued rows and takes on the appearance of a dotted rectangle, regardless of the setting of the [MarqueeStyle](#) property. The behavior of the marquee in other columns is not affected, however.



| Country | First | Last |
|----------------|-----------------|-------------|
| Austria | Alban | Berg |
| | Anton | Bruckner |
| | Gustav | Mahler |
| | Wolfgang Amadeu | Mozart |
| | Arnold | Schoenberg |
| | Franz | Schubert |
| | Anton von | Webern |
| Brazil | Heitor | Villa-Lobos |
| Czechoslovakia | Antonin | Dvorak |
| | Leos | Janacek |
| | Bohuslav | Martinu |
| | Bedrich | Smetana |
| Denmark | Carl | Nielsen |
| England | Benjamin | Britten |
| | Edward | Elgar |

If a design-time layout is specified, the same effect can be achieved by setting the Merge property of the desired [C1DisplayColumn](#) object within the **C1DisplayColumn Collection Editor**, which can be accessed by clicking on the **ellipsis** button (...) after the **DisplayColumns** property in the **Split Collection Editor**.



The Merge property can be set to **Free**, which combines like values in adjacent rows, or **Restricted**, which combines like values in adjacent rows in the same row span as the previous column. The difference between **Free** and

Restricted settings is whether cells within the same contents should always be merged (**Free**) or only when adjacent cells to the left or top are also merged (**Restricted**). The examples below illustrate the difference.

No Merge (Regular Spreadsheet View)

No merge displays data in a regular spreadsheet view.

| Product | Associate | Region | Sales |
|---------|-----------|--------|-------------|
| Drums | Donna | East | \$2,532.00 |
| Drums | Donna | East | \$45,342.00 |
| Drums | John | East | \$14,323.00 |
| Drums | John | North | \$4,543.00 |
| Drums | Paul | North | \$4,232.00 |
| Drums | Paula | East | \$45,342.00 |
| Drums | Sylvia | East | \$45,342.00 |
| Flutes | Donna | South | \$45,342.00 |
| Flutes | John | East | \$43,432.00 |
| Flutes | Paul | North | \$4,543.00 |

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(0).Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.None
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(1).Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.None
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(2).Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.None
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(3).Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.None
```

To write code in C#

C#

```
this.clTrueDBGrid1.Splits[0].DisplayColumns[0].Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.None;
this.clTrueDBGrid1.Splits[0].DisplayColumns[1].Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.None;
this.clTrueDBGrid1.Splits[0].DisplayColumns[2].Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.None;
this.clTrueDBGrid1.Splits[0].DisplayColumns[3].Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.None
```

Free Merge

Free merge combines like values in adjacent rows.

| Product | Associate | Region | Sales |
|---------|-----------|--------|-------------|
| Drums | Donna | East | \$2,532.00 |
| | | | \$45,342.00 |
| | John | | \$14,323.00 |
| | Paul | North | \$4,543.00 |
| | Paula | | \$4,232.00 |
| | Sylvia | East | \$45,342.00 |
| Flutes | Donna | South | |
| | John | East | \$43,432.00 |
| | Paul | North | \$4,543.00 |

Notice how the first *Region* cell (East) merges across employees (Donna and John) to its left.

To write code in Visual Basic

Visual Basic

```
' Set free merging.
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(0).Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.Free
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(1).Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.Free
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(2).Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.Free
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(3).Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.Free

' Set each column's vertical alignment to Center.
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(0).Style.VerticalAlignment =
Cl.Win.ClTrueDBGrid.AlignVertEnum.Center
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(1).Style.VerticalAlignment =
Cl.Win.ClTrueDBGrid.AlignVertEnum.Center
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(2).Style.VerticalAlignment =
Cl.Win.ClTrueDBGrid.AlignVertEnum.Center
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(3).Style.VerticalAlignment =
Cl.Win.ClTrueDBGrid.AlignVertEnum.Center
```

To write code in C#

C#

```
// Set free merging.
this.clTrueDBGrid1.Splits[0].DisplayColumns[0].Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.Free;
this.clTrueDBGrid1.Splits[0].DisplayColumns[1].Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.Free;
this.clTrueDBGrid1.Splits[0].DisplayColumns[2].Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.Free;
this.clTrueDBGrid1.Splits[0].DisplayColumns[3].Merge =
Cl.Win.ClTrueDBGrid.ColumnMergeEnum.Free;

// Set each column's vertical alignment to Center.
this.clTrueDBGrid1.Splits[0].DisplayColumns[0].Style.VerticalAlignment =
Cl.Win.ClTrueDBGrid.AlignVertEnum.Center;
```

```
this.clTrueDBGrid1.Splits[0].DisplayColumns[1].Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center;
this.clTrueDBGrid1.Splits[0].DisplayColumns[2].Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center;
this.clTrueDBGrid1.Splits[0].DisplayColumns[3].Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center;
```

Restricted Merge

Restricted merge combines like values in adjacent rows in the same row span as the previous column.

| Product | Associate | Region | Sales |
|---------|-----------|--------|-------------|
| Drums | Donna | East | \$2,532.00 |
| | | | \$45,342.00 |
| | John | East | \$14,323.00 |
| | | North | \$4,543.00 |
| | Paul | North | \$4,232.00 |
| | Paula | East | \$45,342.00 |
| Flutes | Sylvia | East | \$45,342.00 |
| | Donna | South | \$45,342.00 |
| | John | East | \$43,432.00 |
| | Paul | North | \$4,543.00 |

Notice how the first *Region* cell (East) no longer merges across employees to its left.

To write code in Visual Basic

Visual Basic

```
' Set restricted merging.
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(0).Merge =
C1.Win.C1TrueDBGrid.ColumnMergeEnum.Restricted
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(1).Merge =
C1.Win.C1TrueDBGrid.ColumnMergeEnum.Restricted
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(2).Merge =
C1.Win.C1TrueDBGrid.ColumnMergeEnum.Restricted
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(3).Merge =
C1.Win.C1TrueDBGrid.ColumnMergeEnum.None

' Set each column's vertical alignment to Center.
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(0).Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(1).Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(2).Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center
Me.ClTrueDBGrid1.Splits(0).DisplayColumns(3).Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center
```

To write code in C#

C#

```
// Set restricted merging.
this.clTrueDBGrid1.Splits[0].DisplayColumns[0].Merge =
```


```

C1.Win.C1TrueDBGrid.ColumnMergeEnum.Restricted;
this.c1TrueDBGrid1.Splits[0].DisplayColumns[1].Merge =
C1.Win.C1TrueDBGrid.ColumnMergeEnum.Restricted;
this.c1TrueDBGrid1.Splits[0].DisplayColumns[2].Merge =
C1.Win.C1TrueDBGrid.ColumnMergeEnum.Restricted;
this.c1TrueDBGrid1.Splits[0].DisplayColumns[3].Merge =
C1.Win.C1TrueDBGrid.ColumnMergeEnum.None;

// Set each column's vertical alignment to Center.
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center;
this.c1TrueDBGrid1.Splits[0].DisplayColumns[1].Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center;
this.c1TrueDBGrid1.Splits[0].DisplayColumns[2].Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center;
this.c1TrueDBGrid1.Splits[0].DisplayColumns[3].Style.VerticalAlignment =
C1.Win.C1TrueDBGrid.AlignVertEnum.Center;


```

If the Merge property is set to **Free** or **Restricted** for a column, then none of the data cells can be edited, even if all rows contain unique values. The only exception to this is the AddNew row. However, once a new row is added to the underlying database, then its data will also be uneditable within the merged column(s).

 **Note:** Merged cells are not limited to displaying text. Display bitmaps within merged cells by populating the [ValueItems](#) object as described earlier in [Specifying Text-to-Picture Translations](#). The section [Applying Pictures to Grid Elements](#) describes a more flexible method for displaying in-cell graphics using [Style](#) objects.

Formatting Merged Cells

Use the [HorizontalAlignment](#) and [VerticalAlignment](#) properties of the column's [Style](#) object to center the data within the merged cell, as in the following figure.



| Country | First | Last |
|----------------|-----------------|-------------|
| Austria | Alban | Berg |
| | Anton | Bruckner |
| | Gustav | Mahler |
| | Wolfgang Amadeu | Mozart |
| | Arnold | Schoenberg |
| | Franz | Schubert |
| | Anton von | Webern |
| Brazil | Heitor | Villa-Lobos |
| Czechoslovakia | Antonin | Dvorak |
| | Leos | Janacek |
| | Bohuslav | Martinu |
| | Bedrich | Smetana |
| Denmark | Carl | Nielsen |
| England | Benjamin | Britten |

In the **Splits Collection Editor**, access these properties by expanding the [Style](#) property node at the same level of the tree as the [Merge](#) property. Or, in code:

To write code in Visual Basic

Visual Basic

```
With Me.C1TrueDBGrid1.Splits(0).DisplayColumns("Country").Style
    .HorizontalAlignment = C1.Win.C1TrueDBGrid.AlignHorzEnum.Center
    .VerticalAlignment = C1.Win.C1TrueDBGrid.AlignVertEnum.Center
End With
```

To write code in C#

C#

```
C1.Win.C1TrueDBGrid.Style s;
s = this.c1TrueDBGrid1.Splits[0].DisplayColumns["Country"].Style;
s.HorizontalAlignment = C1.Win.C1TrueDBGrid.AlignHorzEnum.Center;
s.VerticalAlignment = C1.Win.C1TrueDBGrid.AlignVertEnum.Center;
```

Column Grouping

The purpose of this feature is to allow users to dynamically configure a tree view type structure. When in Group mode, a "grouping area" is added to the top of the grid, providing an intuitive interface for specifying column groups. In code, this collection is accessed through the [GroupedColumns](#) collection and consists of [C1DataColumn](#) objects that have been moved to the grouping area; it is similar to the [C1DataColumnCollection](#) class.

The grouping area is created when [DataView](#) is set to [DataViewEnum.GroupBy](#). When [AllowColMove](#) is set to **True**, the grid will support the ability to move one or more columns into this area. Users can do this by selecting a single column and dragging its header into the grouping area. This action can also be performed in code by invoking the [Add](#) method of the [GroupedColumnCollection](#). When a column is first added to the grouping area, nodes are added to the grid. Each node represents the unique value of the grouped column. Similarly when the last grouped column is removed from the area, the nodes are removed and the display will be similar to a normal grid.

When the expand icon ("+") is clicked the grid expands and the next set of grouping column data appears. If there is another grouped column, then this column has an expand icon next to it also. With the addition of each grouped column, another level of sorted information gets added to the tree view. When the expand icon on the final column in the **GroupedColumns** collection is clicked the data in the remaining columns is displayed in the grid's Normal style, as shown below:

| Country | Last | First | Birth | Death |
|-------------------------|------|----------|-------|----------|
| Country: Austria | | | | |
| Country: Brazil | | | | |
| Country: Czechoslovakia | | | | |
| Country: Denmark | | | | |
| Country: England | | | | |
| Country: Finland | | | | |
| Country: France | | | | |
| Country: Janacek | | | | |
| Country: Martinu | | | | |
| Country: Smetana | | | | |
| Country: Dvorak | | | | |
| Antonin | | 9/8/1841 | | 5/1/1904 |

To manipulate the grouping area in code, use the **GroupedColumn** identifiers to access the collection of grouped

columns. Like the **Columns** property, the [GroupedColumns](#) supports [Add](#), and [RemoveAt](#) methods. However, since the [GroupedColumns](#) serves as a placeholder for existing grid columns, the semantics of its [Add](#) and [RemoveAt](#) methods are different.

The [Add](#) method moves an existing column to the grouping area; it does not create a new column in the grid. Similarly, the [RemoveAt](#) method removes a column from the grouping area and returns it to its original position within the grid; it does not delete the column altogether.

Use the [GroupByCaption](#) property to add descriptive or directional text to the grouping area, which will be displayed when no columns are present there.

See [Tutorial 17: Creating a Grouping Display](#) for more information.

Column Grouping with the GroupIntervalEnum Enumeration

The [GroupIntervalEnum](#) enumeration allows you to group data rows according to date, month, year, alphabet, date values (Outlook-style grouping), or you can customize how you would like to sort your data. The following topics explain how to group using a few of these settings.



Note: The default setting is [GroupIntervalEnum.Default](#), which groups rows by their values.

Group Rows by Year

This topic demonstrates how to use the [GroupIntervalEnum.Year](#) member in [C1TrueDBGrid](#).

Complete the following steps:

1. Start a new .NET project.
2. Navigate to the Visual Studio Toolbox and add a [C1TrueDBGrid](#) control to the form.
3. Click the **C1TrueDBGrid** 's smart tag to open the **C1TrueDBGrid Tasks** menu, click the drop-down arrow in the **Choose Data Source** box and choose **Add Project Data Source**.
4. In the **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database.
5. On the **Choose your database objects** page of the wizard, select all fields in the **Employees** table and type "Employees" into the **DataSet name** box, and then finish out the wizard.
6. Visual Studio adds the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.EmployeesTableAdapter.Fill(Me.Employees._Employees)
```

To write code in C#

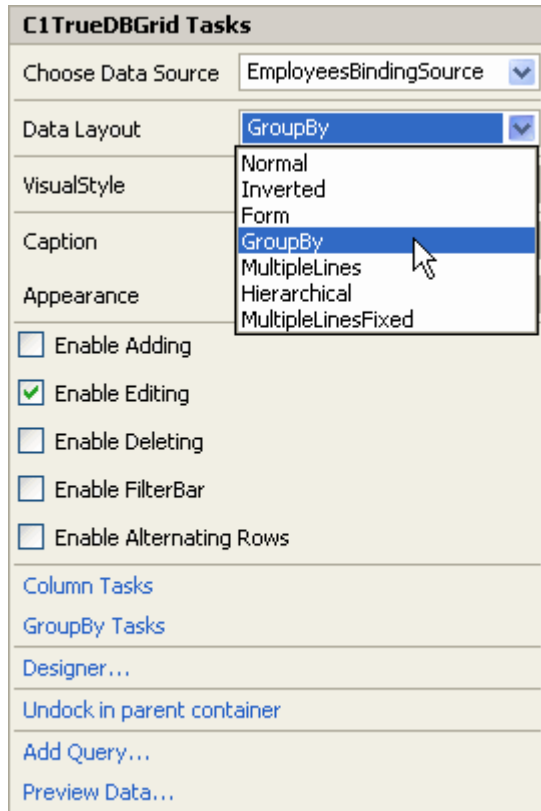
C#

```
this.employeesTableAdapter.Fill(this.Employees._Employees);
```

7. Set the [DataView](#) property to [DataViewEnum.GroupBy](#).

In the Designer

In the **C1TrueDBGrid Tasks** menu, select **GroupBy** from the **Data Layout** drop-down.



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy
```

To write code in C#

C#

```
this.c1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy;
```

8. Open the **C1TrueDBGrid Designer** by selecting **Designer** from the **C1TrueDBGrid Tasks** menu.
9. Select the *HireDate* column by clicking on it in the right pane.



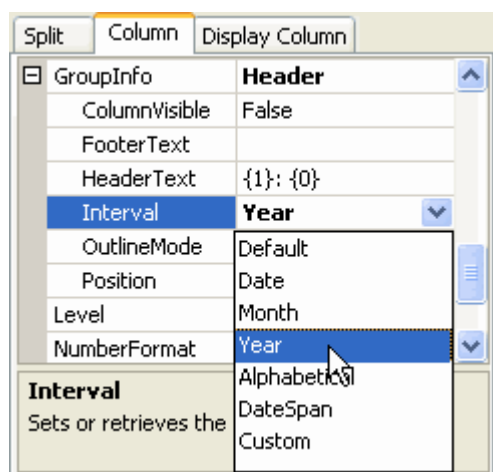
The column can also be selected by choosing *HireDate* from the drop-down list in the toolbar.



10. Set the **Interval** property to **GroupIntervalEnum.Year**.

In the Designer

Locate the **Interval** property in the left pane of the **C1TrueDBGrid Designer** and set it to **Year**.



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' Set the GroupInfo.Interval of the HireDate column to Year.
Me.C1TrueDBGrid1.Columns("HireDate").GroupInfo.Interval =
C1.Win.C1TrueDBGrid.GroupIntervalEnum.Year
```

To write code in C#

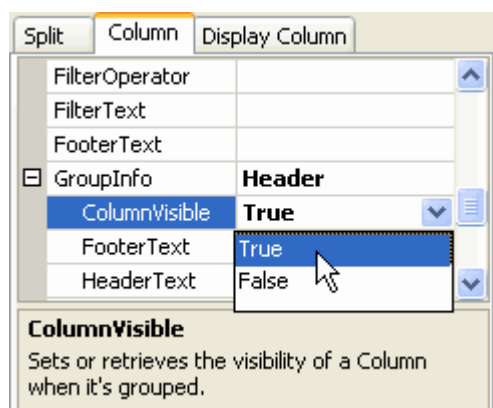
C#

```
// Set the GroupInfo.Interval of the HireDate column to Year.
this.c1TrueDBGrid1.Columns["HireDate"].GroupInfo.Interval =
C1.Win.C1TrueDBGrid.GroupIntervalEnum.Year;
```

- Finally, to keep the *HireDate* column visible after grouping by it, set the **ColumnVisible** property to **True**.

In the Designer

Locate the **ColumnVisible** property in the left pane of the **C1TrueDBGrid Designer** and set it to **True**.



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' Keep the HireDate column visible while grouping.
Me.C1TrueDBGrid1.Columns("HireDate").GroupInfo.ColumnVisible = True
```

To write code in C#

C#

```
// Keep the HireDate column visible while grouping.
this.c1TrueDBGrid1.Columns["HireDate"].GroupInfo.ColumnVisible = true;
```

In this example, the HireDate column is grouped by year.

| HireDate | | | | |
|----------------|------------|-----------|-----------|---------------------|
| | EmployeeID | LastName | FirstName | Title |
| HireDate: 2003 | | | | |
| | 3 | Leverling | Janet | Sales Representati |
| | 1 | Davolio | Nancy | Sales Representati |
| | 2 | Fuller | Andrew | Vice President, Sal |
| HireDate: 2004 | | | | |
| | 4 | Peacock | Margaret | Sales Representati |
| | 5 | Buchanan | Steven | Sales Manager |
| | 6 | Suyama | Michael | Sales Representati |
| HireDate: 2005 | | | | |
| | 7 | King | Robert | Sales Representati |
| | 8 | Callahan | Laura | Inside Sales Coord |
| | 9 | Dodsworth | Anne | Sales Representati |

Group Rows by the First Character of the Value

This topic demonstrates how to use the [GroupIntervalEnum.Alphabetical](#) member in **C1TrueDBGrid**.

Complete the following steps:

1. Start a new .NET project.
2. Open the Toolbox and add a **C1TrueDBGrid** control to the form.
3. Open the **C1TrueDBGrid Tasks** menu, click the drop-down arrow in the **Choose Data Source** box, and click **Add Project Data Source**.
4. In the adapter's **Data Source Configuration Wizard**, either select a connection to C1NWind.mdb or create a new connection to this database.
5. On the **Choose your database objects** page of the wizard, select all fields in the **Products** table and type "Products" into the **DataSet name** box, and then finish out the wizard.
6. Visual Studio adds the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.ProductsTableAdapter.Fill(Me.Products._Products)
```

To write code in C#

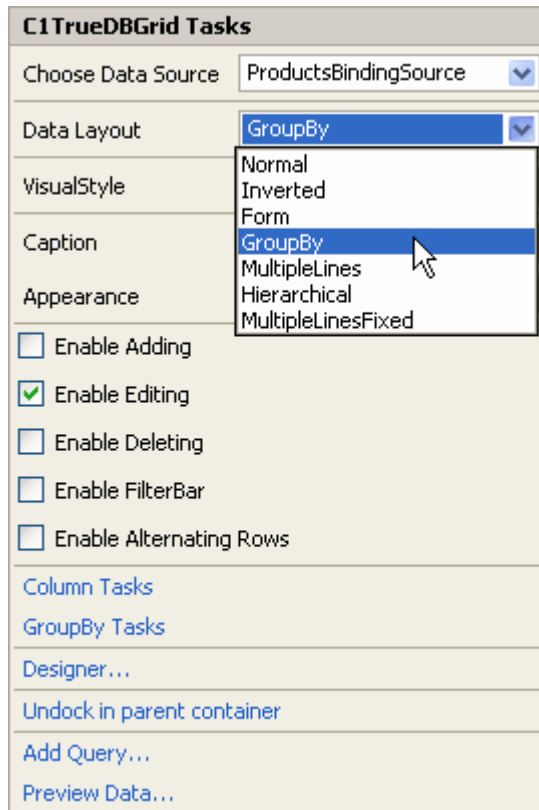
C#

```
this.productsTableAdapter.Fill(this.Products._Products);
```

- Set the **DataView** property to **DataViewEnum.GroupBy**.

In the Designer

In the **C1TrueDBGrid Tasks** menu, select **GroupBy** from the **Data Layout** drop-down.

**In Code**

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

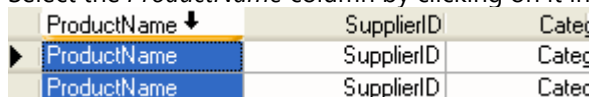
```
Me.C1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy
```

To write code in C#

C#

```
this.c1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy;
```

- Open the **C1TrueDBGrid Designer** by selecting **Designer** from the **C1TrueDBGrid Tasks** menu.
- Select the *ProductName* column by clicking on it in the right pane.



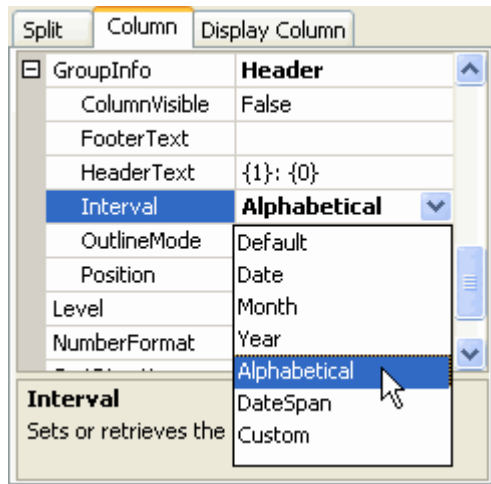
The column can also be selected by choosing *ProductName* from the drop-down list in the toolbar.



10. Set the [Interval](#) property to **Alphabetical**.

In the Designer

Locate the **Interval** property in the left pane of the **C1TrueDBGrid Designer** and set it to **Alphabetical**.



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' Set the GroupInfo.Interval of the ProductName column to Alphabetical.  
Me.C1TrueDBGrid1.Columns("ProductName").GroupInfo.Interval =  
C1.Win.C1TrueDBGrid.GroupIntervalEnum.Alphabetical
```

To write code in C#

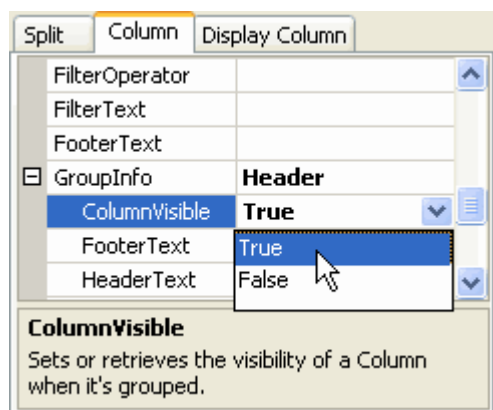
C#

```
// Set the GroupInfo.Interval of the ProductName column to Alphabetical.  
this.C1TrueDBGrid1.Columns["ProductName"].GroupInfo.Interval =  
C1.Win.C1TrueDBGrid.GroupIntervalEnum.Alphabetical;
```

11. Finally, to keep the *ProductName* column visible after grouping by it, set the [ColumnVisible](#) property to **True**.

In the Designer

Locate the **ColumnVisible** property in the left pane of the **C1TrueDBGrid Designer** and set it to **True**.



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' Keep the ProductName column visible while grouping.
Me.C1TrueDBGrid1.Columns("ProductName").GroupInfo.ColumnVisible = True
```

To write code in C#

C#

```
// Keep the ProductName column visible while grouping.
this.c1TrueDBGrid1.Columns["ProductName"].GroupInfo.ColumnVisible = true;
```

In this example, the *ProductName* column is grouped by the first letter of the product name.

| ProductName | CategoryID | Discontinued | ProductID | QuantityPerUnit |
|-------------------|------------|-------------------------------------|-----------|---------------------|
| ▶ ProductName: A | | | | |
| Alice Mutton | 6 | <input checked="" type="checkbox"/> | 17 | 20 - 1 kg tins |
| Aniseed Syrup | 2 | <input type="checkbox"/> | 3 | 12 - 550 ml bottles |
| ▶ ProductName: B | | | | |
| Boston Crab Meat | 8 | <input type="checkbox"/> | 40 | 24 - 4 oz tins |
| ▶ ProductName: C | | | | |
| ▶ ProductName: E | | | | |
| Escargots de Bour | 8 | <input type="checkbox"/> | 58 | 24 pieces |
| ▶ ProductName: F | | | | |
| ▶ ProductName: G | | | | |
| ▶ ProductName: I | | | | |
| ▶ ProductName: J | | | | |
| ▶ ProductName: K | | | | |
| ▶ ProductName: L | | | | |

Group Rows by Date Value (Outlook-Style)

This topic demonstrates how to use the [GroupIntervalEnum.DateSpan](#) member in [C1TrueDBGrid](#).



Note: The C1NWind.mdb database was modified for this example. A field *NextMeeting* was added to the

employees table and filled in with more current dates.

Complete the following steps:

1. Start a new .NET project.
2. Open the Toolbox and add a C1TrueDBGrid control to the form.
3. Open the **C1TrueDBGrid Tasks** menu, click the drop-down arrow in the **Choose Data Source** box, and click **Add Project Data Source**.
4. In the adapter's **Data Source Configuration Wizard**, either select a connection to C1NWind.mdb or create a new connection to this database.
5. On the **Choose your database objects** page of the wizard, select all fields in the **Employees** table and type "Employees" into the **DataSet name** box, and then finish out the wizard.
6. Visual Studio adds the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.EmployeesTableAdapter.Fill(Me.Employees._Employees)
```

To write code in C#

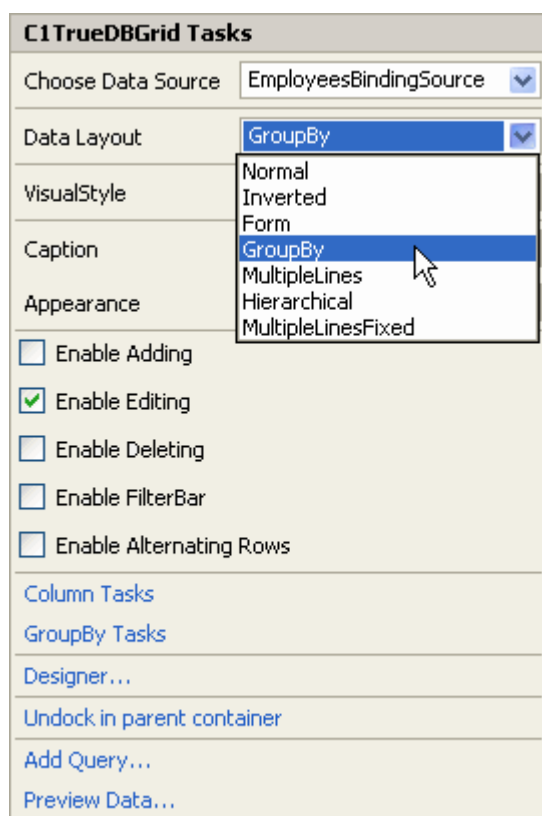
C#

```
this.employeesTableAdapter.Fill(this.Employees._Employees);
```

7. Set the [DataView](#) property to [DataViewEnum.GroupBy](#).

In the Designer

In the **C1TrueDBGrid Tasks** menu, select **GroupBy** from the **Data Layout** drop-down list:



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

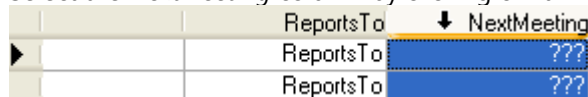
```
Me.C1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy
```

To write code in C#

C#

```
this.C1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy;
```

8. Open the **C1TrueDBGrid Designer** by selecting **Designer** from the **C1TrueDBGrid Tasks** menu.
9. Select the *NextMeeting* column by clicking on it in the right pane.



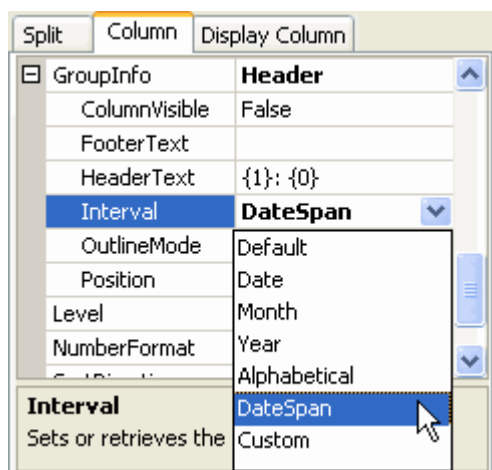
The column can also be selected by choosing *NextMeeting* from the drop-down list in the toolbar.



10. Set the **Interval** property to **GroupIntervalEnum.DateSpan**.

In the Designer

Locate the **Interval** property in the left pane of the **C1TrueDBGrid Designer** and set it to **DateSpan**.



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' Set the GroupInfo.Interval of the grid to DateSpan.
Me.C1TrueDBGrid1.Columns("NextMeeting").GroupInfo.Interval =
C1.Win.C1TrueDBGrid.GroupIntervalEnum.DateSpan
```

To write code in C#

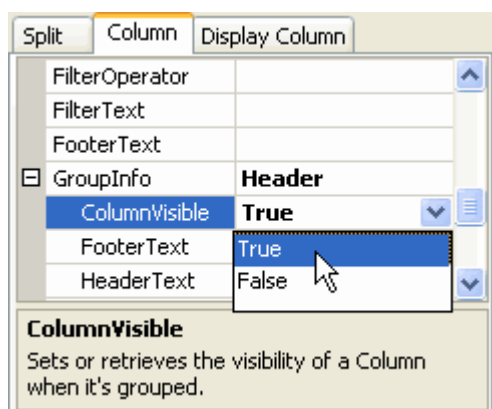
C#

```
// Set the GroupInfo.Interval of the grid to DateSpan.
this.c1TrueDBGrid1.Columns["NextMeeting"].GroupInfo.Interval =
C1.Win.C1TrueDBGrid.GroupIntervalEnum.DateSpan;
```

11. Finally, to keep the *NextMeeting* column visible after grouping by it, set the **ColumnVisible** property to **True**.

In the Designer

Locate the **ColumnVisible** property in the left pane of the **C1TrueDBGrid Designer** and set it to **True**.



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' Keep the NextMeeting column visible while grouping.
Me.C1TrueDBGrid1.Columns("NextMeeting").GroupInfo.ColumnVisible = True
```

To write code in C#

C#

```
// Keep the NextMeeting column visible while grouping.
this.c1TrueDBGrid1.Columns["NextMeeting"].GroupInfo.ColumnVisible = true;
```

In this example, the *NextMeeting* column is sorted by date values.

| NextMeeting / | | | | |
|-----------------------------|------------|-----------|-----------|---------------------|
| | EmployeeID | LastName | FirstName | Title |
| NextMeeting: Last Week | | | | |
| | 1 | Davolio | Nancy | Sales Representati |
| | 2 | Fuller | Andrew | Vice President, Sal |
| NextMeeting: Today | | | | |
| | 9 | Dodsworth | Anne | Sales Representati |
| NextMeeting: Tomorrow | | | | |
| | 8 | Callahan | Laura | Inside Sales Coord |
| NextMeeting: Next Week | | | | |
| | 7 | King | Robert | Sales Representati |
| | 6 | Suyama | Michael | Sales Representati |
| | 4 | Peacock | Margaret | Sales Representati |
| | 3 | Leverling | Janet | Sales Representati |
| NextMeeting: Two Weeks Away | | | | |
| | 5 | Buchanan | Steven | Sales Manager |

Group Rows by Custom Setting

This topic demonstrates how to use the [GroupIntervalEnum.Custom](#) member in **C1TrueDBGrid**.

Complete the following steps:

1. Start a new .NET project.
2. Open the Toolbox and add a **C1TrueDBGrid** control to the form.
3. Open the **C1TrueDBGrid Tasks** menu, click the drop-down arrow in the **Choose Data Source** box, and click **Add Project Data Source**.
4. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database.
5. On the **Choose your database objects** page of the wizard, select all fields in the **Products** table and type "Products" into the **DataSet name** box, and then finish out the wizard.
6. Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.ProductsTableAdapter.Fill(Me.Products._Products)
```

To write code in C#

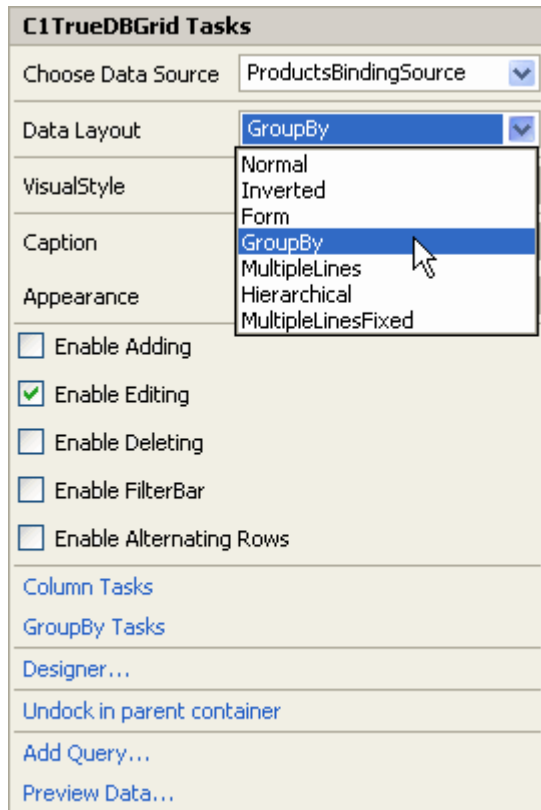
C#

```
this.productsTableAdapter.Fill(this.Products._Products);
```

- Set the **DataGridView** property to **DataGridView.GroupBy**.

In the Designer

In the **C1TrueDBGrid Tasks** menu, select **GroupBy** from the **Data Layout** drop-down.

**In Code**

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

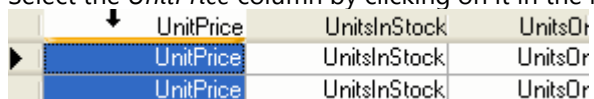
```
Me.C1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy
```

To write code in C#

C#

```
this.c1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy;
```

- Open the **C1TrueDBGrid Designer** by selecting **Designer** from the **C1TrueDBGrid Tasks** menu.
- Select the **UnitPrice** column by clicking on it in the right pane.



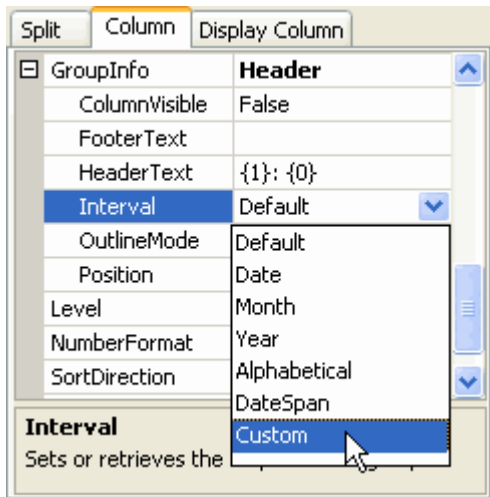
The column can also be selected by choosing *UnitPrice* from the drop-down list in the toolbar.



10. Set the [Interval](#) property to [GroupIntervalEnum.Custom](#).

In the Designer

Locate the [Interval](#) property in the left pane of the **C1TrueDBGrid Designer** and set it to custom.



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' Set the GroupInfo.Interval of the grid to Custom.
Me.C1TrueDBGrid1.Columns("UnitPrice").GroupInfo.Interval =
C1.Win.C1TrueDBGrid.GroupIntervalEnum.Custom
```

To write code in C#

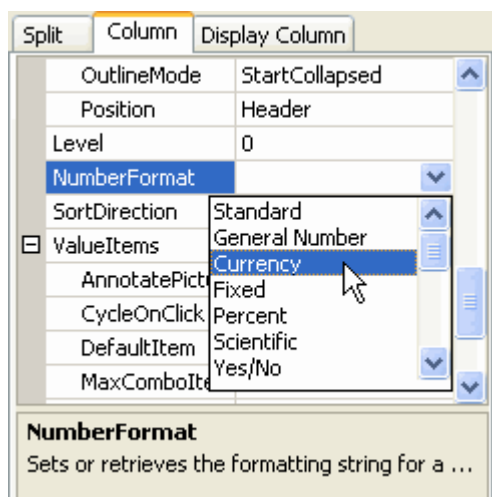
C#

```
// Set the GroupInfo.Interval of the grid to Custom.
this.c1TrueDBGrid1.Columns["UnitPrice"].GroupInfo.Interval =
C1.Win.C1TrueDBGrid.GroupIntervalEnum.Custom;
```

11. Set the [NumberFormat](#) property of the *UnitPrice* column to **Currency**.

In the Designer

Locate the [NumberFormat](#) property in the left pane of the **C1TrueDBGrid Designer** and set it to **Currency**.



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' Set the UnitPrice column to be displayed as currency.  
Me.C1TrueDBGrid1.Columns("UnitPrice").NumberFormat = "Currency"
```

To write code in C#

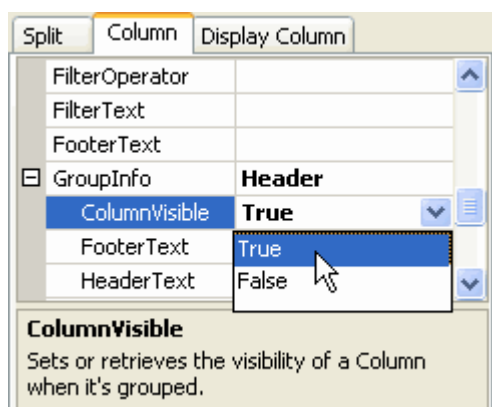
C#

```
// Set the UnitPrice column to be displayed as currency.  
this.c1TrueDBGrid1.Columns["UnitPrice"].NumberFormat = "Currency";
```

12. To keep the *UnitPrice* column visible after grouping by it, set the **ColumnVisible** property to **True**.

In the Designer

Locate the **ColumnVisible** property in the left pane of the **C1TrueDBGrid Designer** and set it to **True**.



In Code

Add the following to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' Keep the UnitPrice column visible while grouping.
Me.ClTrueDBGrid1.Columns("UnitPrice").GroupInfo.ColumnVisible = True
```

To write code in C#

C#

```
// Keep the UnitPrice column visible while grouping.
this.ClTrueDBGrid1.Columns["UnitPrice"].GroupInfo.ColumnVisible = true;
```

13. Add the following code to the [GroupInterval](#) event:

To write code in Visual Basic

Visual Basic

```
' Get the value from the current grid cell as a Decimal.
Dim p As Decimal = CType(Me.ClTrueDBGrid1(e.Row, e.Col.DataColumn.DataField),
Decimal)

' Assign the custom grouping values.
If p > 0 And p < 10 Then
    e.Value = "$0 - $9.99"
ElseIf p >= 10 And p < 20 Then
    e.Value = "$10.00 - $19.99"
ElseIf p >= 20 And p < 40 Then
    e.Value = "$20.00 - $39.99"
ElseIf p >= 40 And p < 60 Then
    e.Value = "$40.00 - $59.99"
ElseIf p >= 60 Then
    e.Value = "$60 And Greater"
End If
```

To write code in C#

C#

```
// Get the value from the current grid cell as a Decimal.
decimal p = ((decimal)this.ClTrueDBGrid1(e.Row,e.Col.DataColumn.DataField));

// Assign the custom grouping values.
If (p > 0 && p < 10)
{
    e.Value = "$0 - $9.99";
}
else if (p >= 10 && p < 20)
{
    e.Value = "$10.00 - $19.99";
}
else if (p >= 20 && p < 40)
{
    e.Value = "$20.00 - $39.99";
}
```

```

else if (p >= 40 && p < 60)
{
    e.Value = "$40.00 - $59.99";
}
else if (p >= 60)
{
    e.Value = "$60 and Greater";
}

```

In this example, the *UnitPrice* column is grouped according to a customized range of values.

| UnitPrice / | | | | | |
|--------------------------------|-------------|------------|-------------------------------------|-----------|--|
| ProductName | UnitPrice / | CategoryID | Discontinued | ProductID | |
| + UnitPrice: \$0 - \$9.99 | | | | | |
| + UnitPrice: \$10.00 - \$19.99 | | | | | |
| + UnitPrice: \$20.00 - \$39.99 | | | | | |
| + UnitPrice: \$40.00 - \$59.99 | | | | | |
| - UnitPrice: \$60 And Greater | | | | | |
| Carnarvon Tigers | \$62.50 | 8 | <input type="checkbox"/> | 18 | |
| Sir Rodney's Marm | \$81.00 | 3 | <input type="checkbox"/> | 20 | |
| Mishi Kobe Niku | \$97.00 | 6 | <input checked="" type="checkbox"/> | 9 | |
| Thüringer Rostbrat | \$123.79 | 6 | <input checked="" type="checkbox"/> | 29 | |
| Côte de Blaye | \$263.50 | 1 | <input type="checkbox"/> | 38 | |

Expanding and Collapsing Grouped Rows

To expand or collapse all grouped rows at once, you can use the [ExpandGroupRow](#) and [CollapseGroupRow](#) methods. In this topic you'll add buttons to your form that will expand and collapse your grouped grid using the [ExpandGroupRow](#) and [CollapseGroupRow](#) methods.

Complete the following steps:

1. Start a new .NET project.
2. Open the Toolbox and add a **SplitContainer** to the form.
3. Select the SplitContainer1's smart tag to open the **SplitContainer Tasks** menu and select **Horizontal splitter orientation**.
4. Select **SplitContainer1.Panel2**, the bottom panel in the SplitContainer and navigate to the Toolbox to add 2 Button controls, Button1 and Button2, to the panel.
5. Resize the buttons on the form, and set the **Text** properties for the buttons in the designer or in code:

In the Designer

In the Properties window set the following properties:

- Select Button1 and in the Properties window set its **Text** property to "Expand".
- Select Button2 and in the Properties window set its **Text** property to "Collapse".

In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.Button1.Text = "Expand"  
Me.Button2.Text = "Collapse"
```

To write code in C#

C#

```
this.button1.Text = "Expand";  
this.button2.Text = "Collapse";
```

6. Select **SplitContainer1.Panel1**, the top panel in the SplitContainer, and navigate to the Toolbox to add a **C1TrueDBGrid** control to the panel.
7. Open the **C1TrueDBGrid Tasks** menu and select **Dock in parent container**.
8. In the **C1TrueDBGrid Tasks** menu, click the drop-down arrow in the **Choose Data Source** box, and click **Add Project Data Source**.
9. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database.
10. On the **Choose your database objects** page of the wizard, select all fields in the **Products** table and type "Products" into the **DataSet name** box, and then finish out the wizard.

Visual Studio adds the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.ProductsTableAdapter.Fill(Me.Products._Products)
```

To write code in C#

C#

```
this.productsTableAdapter.Fill(this.products._Products);
```

11. Set the **DataView** property to **DataViewEnum.GroupBy** in the designer or in code:

In the Designer

In the **C1TrueDBGrid Tasks** menu, select **GroupBy** from the **Data Layout** drop-down.

In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy
```

To write code in C#

C#

```
this.c1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy;
```

12. Add the following **Button_Click** events to the Code Editor to add the **ExpandGroupRow** and **CollapseGroupRow** methods:

To write code in Visual Basic**Visual Basic**

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Me.ClTrueDBGrid1.ExpandGroupRow(-1, True)
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Me.ClTrueDBGrid1.CollapseGroupRow(-1)
End Sub

```

To write code in C#**C#**

```

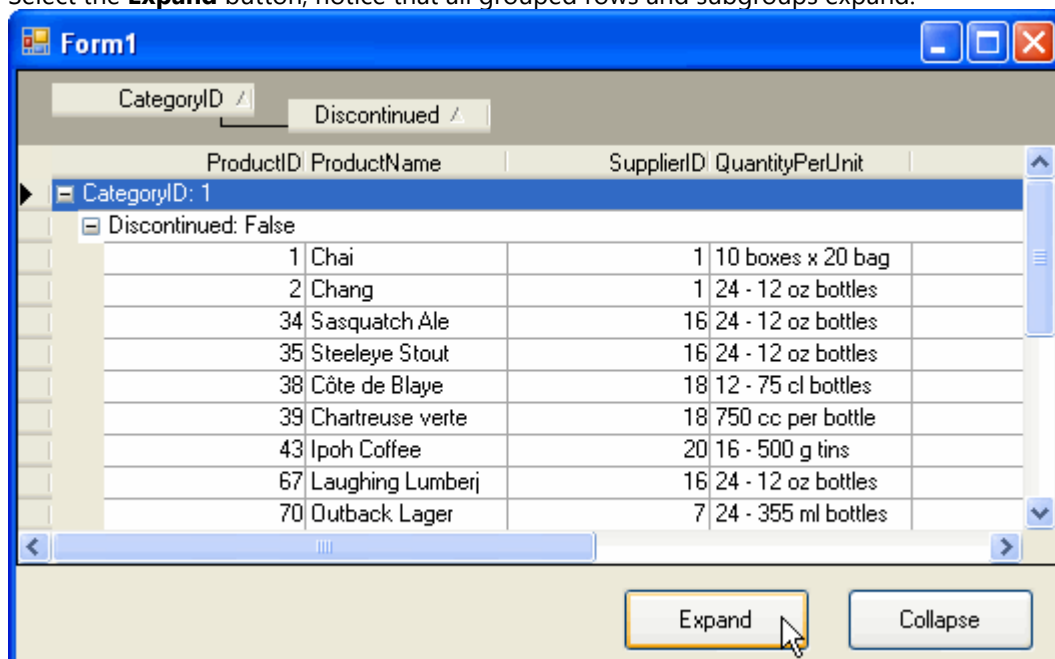
private void button1_Click(object sender, EventArgs e)
{
    this.clTrueDBGrid1.ExpandGroupRow(-1, true);
}

private void button2_Click(object sender, EventArgs e)
{
    this.clTrueDBGrid1.CollapseGroupRow(-1);
}

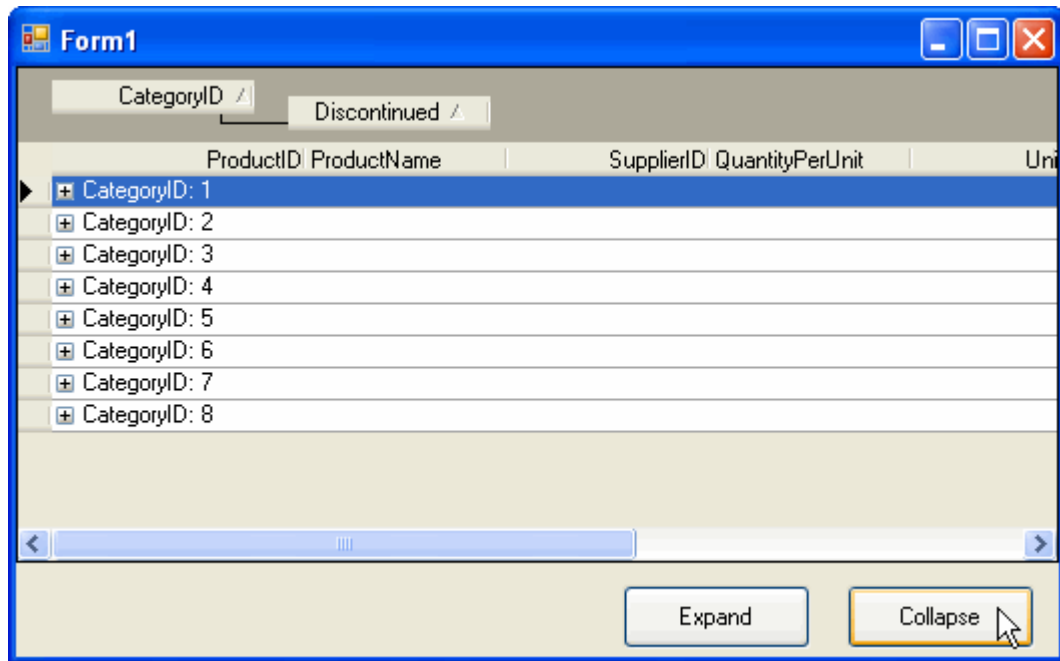
```

Run the application and observe:

1. Group the grid by dragging column headers into the GroupBy area.
2. Select the **Expand** button, notice that all grouped rows and subgroups expand:

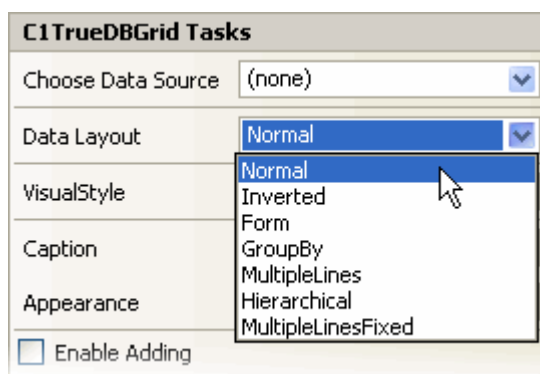


3. Select the **Collapse** button, notice that all grouped rows are now collapsed:



Data Display

True DBGrid for WinForms allows you to view data in different ways through the [DataView](#) property, such as hierarchical, drop-down hierarchical, form, inverted, and multiple line. You can easily change the [DataView](#) property in the Properties window, in code, or by selecting a **Data Layout** option in the **C1TrueDBGrid Tasks** menu:



The following topics describe the different data views available in the [C1TrueDBGrid](#) control.

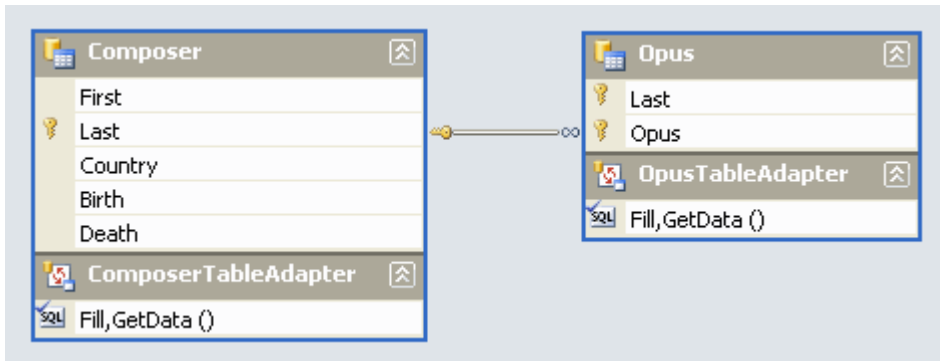
Hierarchical Data Display

True DBGrid for WinForms supports the ability to display hierarchical data. Hierarchical data generally refers to data that is stored in multiple relational tables, where a master (or "parent") table is linked by keyed fields to detail (or "child") tables. The hierarchical display provides the ability to present the master data to users, such that the related detail data can be viewed in the same grid with a single mouse click.

When the grid is bound to a master-detail data source, display related groups of hierarchical data by using bands. A band is a virtual representation of a hierarchical DataSet, not the data itself. A band is created for each level in a hierarchical recordset, and may consist of entire tables or a just a few selected fields. At run time, users can expand and collapse bands using a TreeView-like interface.

To use this feature, the [DataView](#) property must be set to [DataViewEnum.Hierarchical](#). The grid control must be bound to a hierarchical DataSet. One way to do this is to use the [DataSource](#) property.

In this example there is a relation between the **Composer** and **Opus** tables. Both tables have a *Last* field, which happens to be the primary key for the table. The *Last* field of both **Composer** and **Opus** are identical. Thus when joined together on this field these two tables create a hierarchical DataSet.



This hierarchical DataSet can be displayed in the grid through the use of bands and the grid's hierarchical display. By completing just three steps, the above DataSet can be displayed in the **C1TrueDBGrid** control. These steps are:

1. First the [DataSource](#) property of the grid needs to be set to the hierarchical DataSet.
2. Secondly, the [DataMember](#) property of the grid needs to be set to the parent table in the DataSet. This will tell the grid which table must be displayed initially. In this example, the parent table is **Composer**.
3. Finally, the grid needs to know to switch to the hierarchical display. By setting the [DataView](#) property to [DataViewEnum.Hierarchical](#), the grid will display the above dataset with its bands structure.

At run time, the grid displays read-only data. The next figure illustrates the initial display of the grid. The data from the master recordset (**Composer**) is displayed first, and the fields from the detail recordset bands appear to the right. The detail recordset fields initially contain no data, however. An expand icon ("+") at the left edge of a record indicates the presence of hierarchical data.

| Composers | | | | | | |
|-----------|------------------|---------------|------------|-----------|--------|--------------------|
| Last | First | Country | Birth | Death | Last | Opus |
| Albeniz | Isaac | Spain | 5/29/1860 | 5/18/1909 | | |
| Bach | Johann Sebastian | Germany | | | | |
| Barber | Samuel | United States | 3/9/1910 | | Barber | Adagio for Strings |
| | | | | | Barber | Medea's Meditation |
| | | | | | Barber | Overture to "The S |
| | | | | | Barber | Second Essay for |
| Bartok | Bela | Hungary | 3/25/1881 | 9/26/1945 | | |
| Beethoven | Ludwig van | Germany | 12/16/1770 | 3/26/1827 | | |

When the user clicks an expand icon, it changes to a collapse icon ("–") and the next band (**Opus**) expands to show the detail records for the clicked row.

Note: If the [DataView](#) property is set to its default value of [DataViewEnum.Normal](#), the grid will only display flat files; it will not support a hierarchical view. Even if the data source is a hierarchical DataSet, the grid will only display data from the master table.

The [DataView](#) property must be set at design time; it cannot be changed at run time.

The following methods are provided for manipulating hierarchical grid displays:

| Method | Description |
|------------------------------|---|
| GetBand | Returns the band for a specified column index. |
| CollapseBand | Collapses all rows for the specified band. |
| ExpandBand | Expands all rows for the specified band. |
| RowExpanded | Returns True if the current row is expanded within the specified band. |

If the number of recordset levels in a master-detail data source is not known in advance, examine the [Bands](#) property in code. Allowable band numbers range from 0 to [Bands](#) - 1.

The following events enable the application to respond to hierarchical view operation initiated by the user:

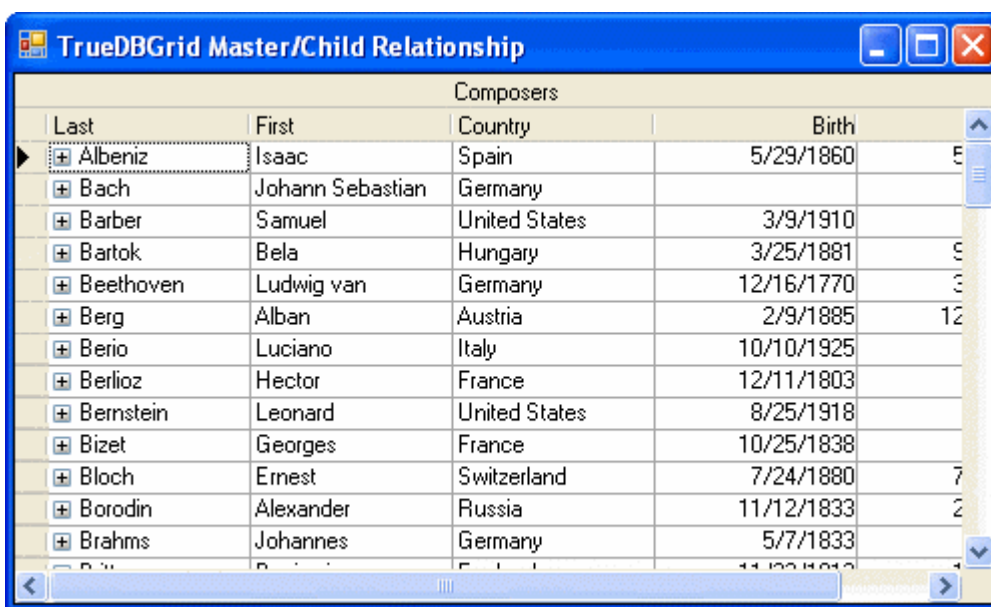
| Event | Description |
|--------------------------|---|
| Collapse | Fired when a band is collapsed by a user. |
| Expand | Fired when a band is expanded by a user. |

Drop-Down Hierarchical Data Display

True DBGrid for WinForms allows you to display a master/child relationship between data sources in such a way that the child data records are available from within the master table in a completely new **True DBGrid**. By simply setting the [ChildGrid](#) property to connect two grid controls and a few lines of code, you can create a fully editable drop-down child that appears within the master table with a simple click.

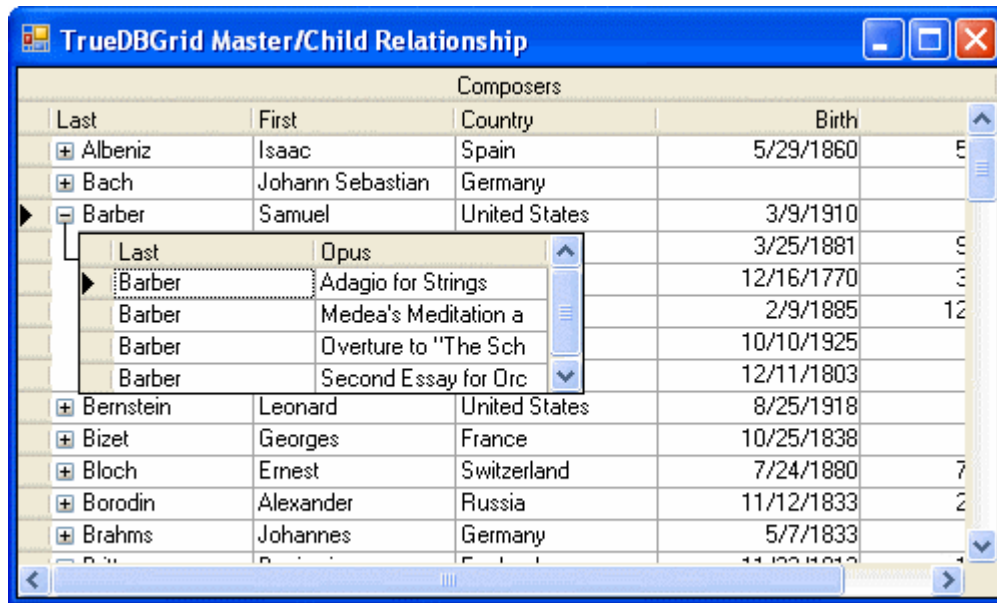
Assuming that your hierarchical dataset is already configured, you can create the master/child relationship by selecting **C1TrueDBGrid2** in the [ChildGrid](#) property of **C1TrueDBGrid1**.

Notice that **C1TrueDBGrid2** is rendered invisible and there is an expand icon ("+") beside the left most cell in each row. The master table contains a list of composers including vital statistics. Note, that as you scroll right, the expand icon remains in the left most cell at all times.



By left clicking on any of the expand icons, our child table appears in a drop-down window. In this case, the drop-

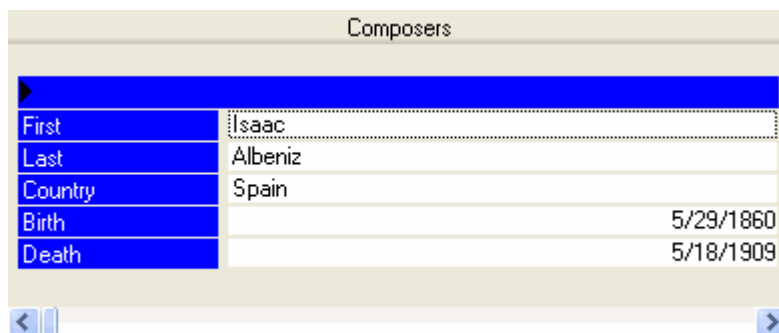
down window lists the written works of the specific composer that you expanded.



This demonstrates how very simple it is to attach a child grid to its master grid and display hierarchical data in a convenient display.

Form Data Display

In situations where you would like to display data one record at a time, you can set the [DataView](#) property to [DataViewEnum.Form](#). You can set this property either in the designer or in code to display data in an editable form similar to the one in the following illustration.



To adjust the width of the data column area or the caption column area, change the [ViewColumnWidth](#) and [ViewCaptionWidth](#) properties to create the appropriate column spacing.

Inverted Data Display

The inverted option of the [DataView](#) property inverts each row in your data into columns. In effect, the leftmost column becomes the top row, the second column becomes the second row, and so forth. Use this display to maximize screen real estate for tables that have many columns. Set the [DataView](#) property to [DataViewEnum.Inverted](#) to display an inverted grid as depicted in the following illustration.

| Composers | | | | |
|-----------|-----------|------------------|---------------|---------|
| First | Isaac | Johann Sebastian | Samuel | Bela |
| Last | Albeniz | Bach | Barber | Bartok |
| Country | Spain | Germany | United States | Hungary |
| Birth | 5/29/1860 | | 3/9/1910 | 3/25/1 |
| Death | 5/18/1909 | | | 9/26/1 |

To adjust the width of the data column area or the caption column area, you can change the [ViewColumnWidth](#) and [ViewCaptionWidth](#) properties to create the appropriate column spacing.

Multiple Line Data Display

Normally, a record is displayed in a single row in the grid. If the grid is not wide enough to display all of the columns in the record, a horizontal scroll bar automatically appears to enable users to scroll columns in and out of view. For discussion purposes, the following will be distinguished:

- A **line** in a grid is a single *physical row* of cells displayed in the grid. Do not confuse this with a line of text inside a grid cell; depending upon the settings of the [RowHeight](#) and [WrapText](#) properties, data in a grid cell may be displayed in multiple lines of text.
- A **row** in a grid is used to display a single record. A row may contain multiple lines or multiple *physical rows*.

Setting the [DataView](#) property to [DataViewEnum.MultipleLines](#) will display every field of data in the dataset in the available grid area. If the dataset contains more fields than can fit in the grid area, then a single record will span multiple lines. This enables the end user to simultaneously view all of the columns (fields) of a record within the width of the grid without scrolling horizontally:

| Composers | | | |
|------------------|-----------|---------------|--|
| First | Last | Country | |
| Birth | Death | | |
| Isaac | Albeniz | Spain | |
| | 5/29/1860 | 5/18/1909 | |
| Johann Sebastian | Bach | Germany | |
| | | | |
| Samuel | Barber | United States | |
| | 3/9/1910 | | |
| Bela | Bartok | Hungary | |

You can adjust resulting column layout at either design time or run time by changing the widths and orders of the columns. When changing the width of a column, the grid will only increase the size of the column at the expense of the other columns in the line. Unlike previous versions of the grid, the columns will not wrap to another line if a column is resized.

To change the order of the columns while in MultipleLine view, click and drag the column header to the new position. A red arrow should indicate where the column is to be placed. After the column has been dropped, the grid will reposition the columns accordingly.

Note that you can specify a multiple line mode in which the grid does scroll horizontally by setting the [DataView](#) property to [DataViewEnum.MultipleLinesFixed](#), see [Multiple Line Fixed Data Display](#) for more information.



Note: At design time, if the [HScrollBar](#) and [VScrollBar](#) style property is set to [ScrollBarStyleEnumAutomatic](#), and the [DataView](#) property is set to [DataViewEnum.MultipleLines](#), a vertical scroll bar appears even though no data

is displayed. This is done so the width of the scroll bar can be taken into account when adjusting columns at design time.

Implications of Multiple-Line Mode

Existing row-related properties, methods, and events fit well with the earlier definitions of records, rows, and lines (with two exceptions to be described later). For example:

- The [VisibleRows](#) property returns the number of visible rows or records displayed on the grid—not the number of visible lines. If a row spans 2 lines, and the [VisibleRows](#) property is 5, then there are 10 visible lines displayed on the grid.
- The [RowTop](#) method accepts a row number argument ranging from 0 to [VisibleRows](#) - 1. If a row spans 2 lines, then [RowTop](#) returns the position of the top of the third displayed row (that is, the fifth displayed line).
- The [RowResize](#) event will be fired whenever a row is resized by the user at run time. In fact, at the record selector column, only row divider boundaries are displayed; thus, the user can only resize rows, not lines.

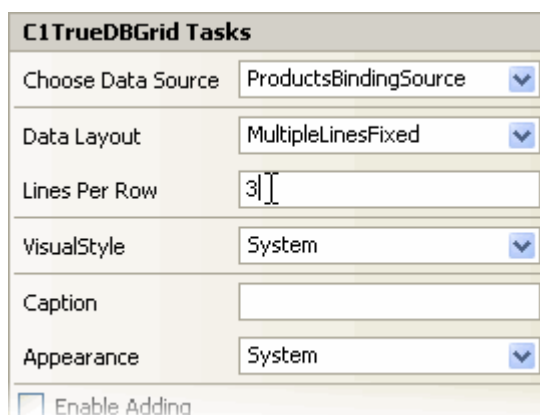
Other row-related properties, methods, and events can be interpreted similarly. There are two exceptions:

1. The first is the [RowHeight](#) property. The [RowHeight](#) property returns the height of a cell or a line, **not** the height of a row. Changing this property would break users' existing code.
2. The second is more of a limitation than an exception. Currently the dividers between rows and lines are the same. When the [RowDivider](#) object's style property is changed, all dividers between rows and lines change to the same style. That is, different dividers cannot exist for rows and for lines.

Multiple Line Fixed Data Display

Setting the [DataView](#) property to [DataViewEnum.MultipleLines](#) enables the end user to simultaneously view all of the columns (fields) of a record within the width of the grid without scrolling horizontally. But if you want to have more control over the multiple line data view, including visible horizontal scroll bars, you can set the [DataView](#) property to [DataViewEnum.MultipleLinesFixed](#) instead.

The [DataViewEnum.MultipleLinesFixed](#) data view is very similar to the [DataViewEnum.MultipleLines](#) data view but the number of subrows does not change once set. The number of subrows can be set using the [LinesPerRow](#) property which can be set at code or in the **C1TrueDBGrid Tasks** menu:



The screenshot shows the 'C1TrueDBGrid Tasks' menu with the following settings:

| C1TrueDBGrid Tasks | |
|--|-----------------------|
| Choose Data Source | ProductsBindingSource |
| Data Layout | MultipleLinesFixed |
| Lines Per Row | 3 |
| VisualStyle | System |
| Caption | |
| Appearance | System |
| <input type="checkbox"/> Enable Adding | |

Row widths in this [DataView](#) are not constrained by the width of the grid; if the sub of the column width is greater than the client width of the grid you will now get a horizontal scrollbar:

| ProductID | ProductName | SupplierID | CategoryID |
|---------------------|---------------|--------------|--------------------------|
| QuantityPerUnit | UnitPrice | UnitsInStock | UnitsOnOrd |
| ReorderLevel | Discontinued | | |
| 1 | Chai | | 1 |
| 10 boxes x 20 bag | | 18 | 39 |
| | | 10 | <input type="checkbox"/> |
| 2 | Chang | | 1 |
| 24 - 12 oz bottles | | 19 | 17 |
| | | 25 | <input type="checkbox"/> |
| 3 | Aniseed Syrup | | 1 |
| 12 - 550 ml bottles | | 10 | 13 |
| | | 25 | <input type="checkbox"/> |

Note that you can also merge the left-most column. Setting the [Merge](#) property allows the left-most column to span the height of the row. For example in the image below, the *ProductID* column spans the row:

| ProductID | ProductName | SupplierID | CategoryID |
|---------------------|---------------|--------------|------------|
| QuantityPerUnit | UnitPrice | UnitsInStock | UnitsOnOrd |
| ReorderLevel | Discontinue | | |
| 1 | Chai | | 1 |
| 10 boxes x 20 bag | | 18 | |
| | | 10 | |
| 2 | Chang | | 1 |
| 24 - 12 oz bottles | | 19 | |
| | | 25 | |
| 3 | Aniseed Syrup | | 1 |
| 12 - 550 ml bottles | | 10 | |
| | | 25 | |

Note: The merge property is only applicable for the left-most columns in the grid.

Owner-Drawn Cells

For cases where complex per-cell customizations need to be performed you can render the contents of the cell by writing a handler for the [OwnerDrawCell](#) event. This event is raised as needed to display the contents of cells that have their [OwnerDraw](#) property set to **True**.



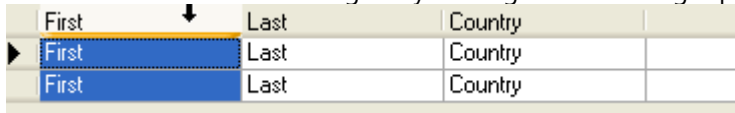
| First | Last | Country | Birth | |
|------------------|-----------|---------------|------------|----|
| Isaac | Albeniz | Spain | 5/29/1860 | 5 |
| Johann Sebastian | Bach | Germany | | |
| Samuel | Barber | United States | 3/9/1910 | |
| Bela | Bartok | Hungary | 3/25/1881 | 9 |
| Ludwig van | Beethoven | Germany | 12/16/1770 | 3 |
| Alban | Berg | Austria | 2/9/1885 | 12 |
| Luciano | Berio | Italy | 10/10/1925 | |
| Hector | Berlioz | France | 12/11/1803 | |
| Leonard | Bernstein | United States | 8/25/1918 | |
| Georges | Bizet | France | 10/25/1838 | |
| Ernest | Bloch | Switzerland | 7/24/1880 | 7 |
| Alexander | Borodin | Russia | 11/12/1833 | 2 |
| Johannes | Brahms | Germany | 5/7/1833 | |
| Benjamin | Britten | England | 11/22/1913 | 1 |
| Max | Buck | Germany | 1/10/1870 | 1 |

To create the owner-drawn cells in the above illustration, complete the following:

1. Set the **OwnerDraw** property to **True** for the *First* column either in the designer or in code:

In the Designer

- Open the **C1TrueDBGrid Designer** by selecting **Designer** from the **C1TrueDBGrid Tasks** menu.
- Select the *First* column in the grid by clicking on it in the right pane.

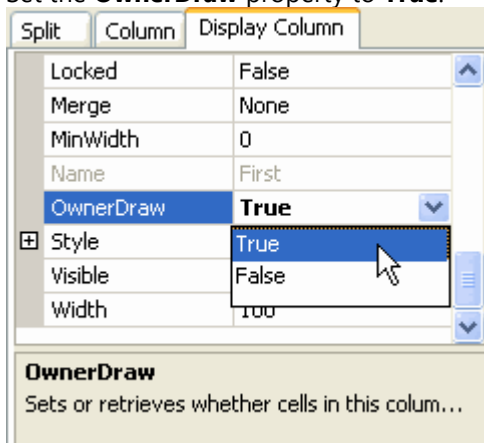


| First | Last | Country | |
|-------|------|---------|--|
| First | Last | Country | |
| First | Last | Country | |

The column can also be selected by choosing *First* from the drop-down list in the toolbar.



- Click the **Display Column** tab in the left pane.
- Set the **OwnerDraw** property to **True**.



| Split | Column | Display Column |
|-----------|--------|----------------|
| Locked | False | |
| Merge | None | |
| MinWidth | 0 | |
| Name | First | |
| OwnerDraw | True | |
| Style | True | |
| Visible | False | |
| Width | 100 | |

OwnerDraw
Sets or retrieves whether cells in this colum...

- Click **OK** to close the editor.

In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns("First").OwnerDraw = True
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].DisplayColumns["First"].OwnerDraw = true;
```

2. Declare the structure RECT in the general declarations of the form:

To write code in Visual Basic

Visual Basic

```
Public Structure RECT
    Dim Left As Long
    Dim Top As Long
    Dim Right As Long
    Dim Bottom As Long
End Structure
```

To write code in C#

C#

```
public struct RECT{
    long Left;
    long Top;
    long Right;
    long Bottom;
}
```

3. Implement the **OwnerDrawCell** event as follows:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_OwnerDrawCell(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.OwnerDrawCellEventArgs) Handles C1TrueDBGrid1.OwnerDrawCell
    If e.Col = 0 Then

        ' Create a gradient brush, blue to red.
        Dim pt1, pt2 As Point
        pt1 = New Point(e.CellRect.X, e.CellRect.Y)
        pt2 = New Point(e.CellRect.Right, e.CellRect.Y)
        Dim linGrBrush As System.Drawing.Drawing2D.LinearGradientBrush
        linGrBrush = New System.Drawing.Drawing2D.LinearGradientBrush(pt1, pt2,
Color.Blue, Color.Red)

        Dim rt As RectangleF
        rt = New RectangleF(e.CellRect.X, e.CellRect.Y, e.CellRect.Width,
e.CellRect.Height)

        ' Fill the cell rectangle with the gradient.
```

```
e.Graphics.FillRectangle(linGrBrush, e.CellRect)

Dim whiteBR As Brush
whiteBR = New SolidBrush(Color.White)
Dim dispCol As C1.Win.C1TrueDBGrid.C1DisplayColumn
dispCol = Me.C1TrueDBGrid1.Splits(0).DisplayColumns(e.Col)

' Center the text horizontally.
Dim sfmt As New StringFormat()
sfmt.Alignment = StringAlignment.Center

' Draw the text.
e.Graphics.DrawString(dispCol.DataColumn.CellText(e.Row),
dispCol.Style.Font, whiteBR, rt, sfmt)
whiteBR.Dispose()

' Let the grid know the event was handled.
e.Handled = True
End If
End Sub
```

To write code in C#

C#

```
private void C1TrueDBGrid1_OwnerDrawCell(object sender,
C1.Win.C1TrueDBGrid.OwnerDrawCellEventArgs e)
{
    if ( e.Col = 0 )
    {
        // Create a gradient brush, blue to red.
        Point pt1, pt2;
        pt1 = new Point[e.CellRect.X, e.CellRect.Y];
        pt2 = new Point[e.CellRect.Right, e.CellRect.Y];
        System.Drawing.Drawing2D.LinearGradientBrush linGrBrush;
        linGrBrush = new System.Drawing.Drawing2D.LinearGradientBrush(pt1, pt2,
Color.Blue, Color.Red);

        RectangleF rt;
        rt = new RectangleF(e.CellRect.X, e.CellRect.Y, e.CellRect.Width,
e.CellRect.Height);

        // Fill the cell rectangle with the gradient.
        e.Graphics.FillRectangle(linGrBrush, e.CellRect);

        Brush whiteBR;
        whiteBR = new SolidBrush(Color.White);
        C1.Win.C1TrueDBGrid.C1DisplayColumn dispCol;
        dispCol = this.c1TrueDBGrid1.Splits[0].DisplayColumns[e.Col];

        // Center the text horizontally.
        StringFormat sfmt = new StringFormat();
```

```
        sfmt.Alignment = StringAlignment.Center;

        // Draw the text.
        e.Graphics.DrawString(dispCol.DataColumn.CellText[e.Row],
dispCol.Style.Font, whiteBR, rt, sfmt);
        whiteBR.Dispose();

        // Let the grid know the event was handled.
        e.Handled = true;
    }
}
```

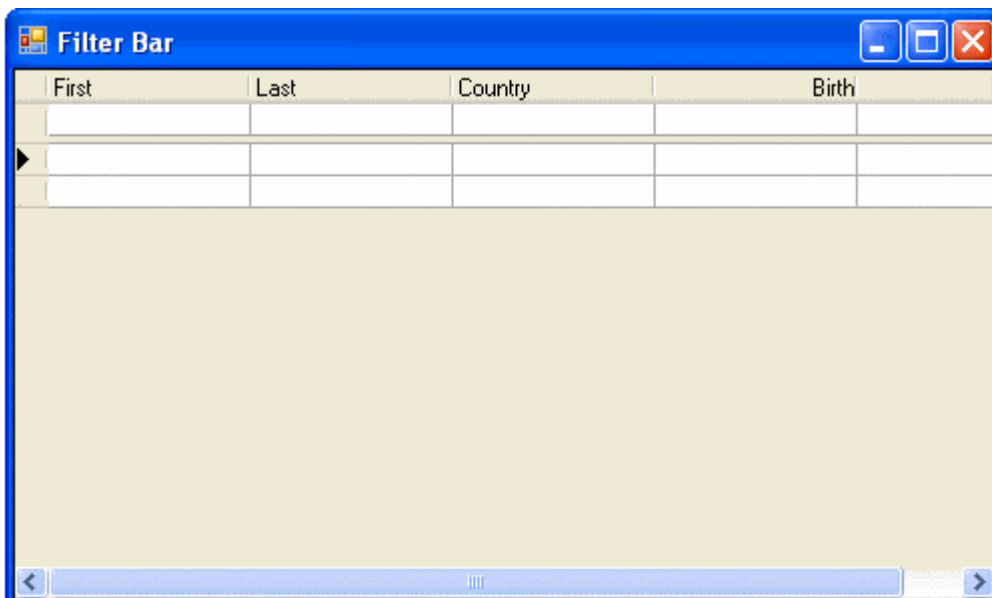
There are a couple key points worth noting in this example:

- If the [Handled](#) property is set to **True**, the grid will **not** fill in the cell's background, nor will it display cell text or graphics. Therefore, you are responsible for filling in the **entire cell**, even if there is no data to display.
- Even a relatively simple example such as the one illustrated here requires a fair amount of coding, so consider using background bitmaps instead of owner-drawn cells if possible.

Filtering Data in DataSets

In some cases, you might want to allow users to filter the underlying recordset at run time by limiting the number of items in a given field or fields. By using the [FilterBar](#) and [AllowFilter](#) properties at design time, and entering the filter text appropriately at run time, the number of field entries can be reduced almost effortlessly.

When the [FilterBar](#) property of a [C1TrueDBGrid](#) control is set to **True**, a blank row with a gray separator line appears directly above the uppermost data row in the grid:



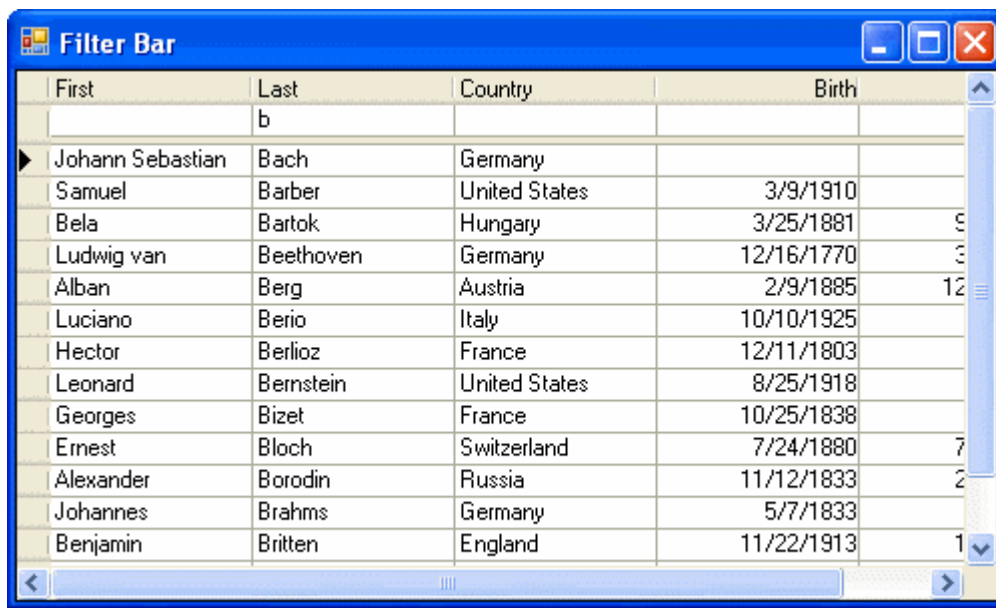
In order to implement the filter in the grid, the [AllowFilter](#) property must be set to **True** (default), which will tell the grid to implement the filtering process.

If the [FilterBar](#) and [AllowFilter](#) properties are both set to **True**, the filter bar will appear in the grid and the grid will also handle automatically the handling of the DataSet.

Manually Filtering Data

In the event that you would prefer to handle the filtering process yourself, leaving the [AllowFilter](#) property as **False** will not implement the grid's automatic filter. In order to create a filter, the [FilterChange](#) event, must be used to manually sort the data. This event fires whenever there the user changes the state of the filter bar.

In this event, a handler would have to be created which filters the dataset for each character the user enters. For example, if the user types "B" in a filter bar cell, the underlying dataset would have to be limited to just those column items whose values start with the letter B. If the user then extended the filter to "BR", then the list would have to be reduced to only those whose values that start with BR.



Adding a Watermark to the Filter Bar

You can now easily add a text watermark to the filter bar so that it appears with default text. You can use this watermark to add instructions for filtering text, or adding default values to give users a better understanding of what values can be entered in particular filter bar cells. All you need to do to have text appear in the filter bar is set the [FilterWatermark](#) property to a string.

For example in the following code, the [FilterWatermark](#) in the first filter bar cell is set to "Filter Me":

To write code in Visual Basic

Visual Basic

```
' Set the C1DataColumn.FilterWatermark property of the first column.
Me.C1TrueDBGrid1.Columns(0).FilterWatermark = "Filter Me"
```

To write code in C#

C#

```
// Set the C1DataColumn.FilterWatermark property of the first column.
this.c1TrueDBGrid1.Columns[0].FilterWatermark = "Filter Me";
```

Notice that the background color of the filter bar cell with a watermark has changed:



| First | Last | Country | |
|------------------|-----------|---------------|--------|
| Filter Me | | | |
| Isaac | Albeniz | Spain | 5/29/ |
| Johann Sebastian | Bach | Germany | |
| Samuel | Barber | United States | 3/9/ |
| Bela | Bartok | Hungary | 3/25/ |
| Ludwig van | Beethoven | Germany | 12/16/ |
| Alban | Berg | Austria | 2/9/ |
| Luciano | Berio | Italy | 10/10/ |
| Hector | Berlioz | France | 12/11/ |
| Leonard | Bernstein | United States | 8/25/ |
| Georges | Bizet | France | 10/25/ |

In the following code, the `FilterWatermark` is set to the value of each column's caption text:

To write code in Visual Basic

Visual Basic

```
Dim colcount As Integer
colcount = 0
While C1TrueDBGrid1.Columns.Count > colcount
    ' Set the C1DataColumn.FilterWatermark property of each column to its caption.
    C1TrueDBGrid1.Columns(colcount).FilterWatermark =
    C1TrueDBGrid1.Columns(colcount).Caption
    colcount = colcount + 1
End While
```

To write code in C#

C#

```
int colcount;
colcount = 0;
while (c1TrueDBGrid1.Columns.Count > colcount)
{
    // Set the C1DataColumn.FilterWatermark property of each column to its caption.
    this.c1TrueDBGrid1.Columns[colcount].FilterWatermark =
    c1TrueDBGrid1.Columns[colcount].Caption;
    colcount = colcount + 1;
}
```

The grid appears like the following:

| First | Last | Country | |
|------------------|-----------|---------------|--------|
| First | Last | Country | Birth |
| Isaac | Albeniz | Spain | 5/29/ |
| Johann Sebastian | Bach | Germany | |
| Samuel | Barber | United States | 3/9/ |
| Bela | Bartok | Hungary | 3/25/ |
| Ludwig van | Beethoven | Germany | 12/16/ |
| Alban | Berg | Austria | 2/9/ |
| Luciano | Berio | Italy | 10/10/ |
| Hector | Berlioz | France | 12/11/ |
| Leonard | Bernstein | United States | 8/25/ |
| Georges | Bizet | France | 10/25/ |

Notice that when text is filtered, the watermark is no longer visible:

| First | Last | Country | Birth |
|------------------|--------|---------------|----------|
| First | ba | Country | Birth |
| Johann Sebastian | Bach | Germany | |
| Samuel | Barber | United States | 3/9/191 |
| Bela | Bartok | Hungary | 3/25/188 |

You can change the appearance of the style of the [FilterWatermark](#) using the [FilterWatermarkStyle](#) property. See [How to Use Styles](#) for more information about styles.

Filtering the Grid with Multiple Criteria

You can now easily filter the grid with multiple filter criteria at run time. For example, you can filter the grid so that only items starting with the letter A or the letter B appear in the grid (instead of limiting the filter to one or the other). All you need to do to have text appear in the filter bar is set the [FilterMultiSelect](#) property to **True**.

For example in the following code, the [FilterMultiSelect](#) property in the first filter bar cell is set to **True**:

To write code in Visual Basic

Visual Basic

```
' Display the filter bar.
Me.C1TrueDBGrid1.FilterBar = True
' Allow the first column to be filtered by multiple items.
Me.C1TrueDBGrid1.Columns(0).FilterMultiSelect = True
```

To write code in C#

C#

```
// Display the filter bar.
this.C1TrueDBGrid1.FilterBar = true;
```

```
// Allow the first column to be filtered by multiple items.
this.clTrueDBGrid1.Columns[0].FilterMultiSelect = true;
```

If you run the application, you'll notice that you can filter the first cell with multiple criteria. For example type "a,b" in the filter bar and notice that items starting with the letter A and the letter B are displayed. You can customize the character used for separating filter items by setting the [FilterSeparator](#) property.

Adding a Filter Drop-Down List

In addition to the filter bar, you can also include a drop-down filter list in the filter bar. The drop-down list lists every item in that column and provides a simple way that users can choose what items to filter the column by without entering their own value.

For example in the following code, the [FilterDropDown](#) property in the second filter bar cell is set to **True**:

To write code in Visual Basic

Visual Basic

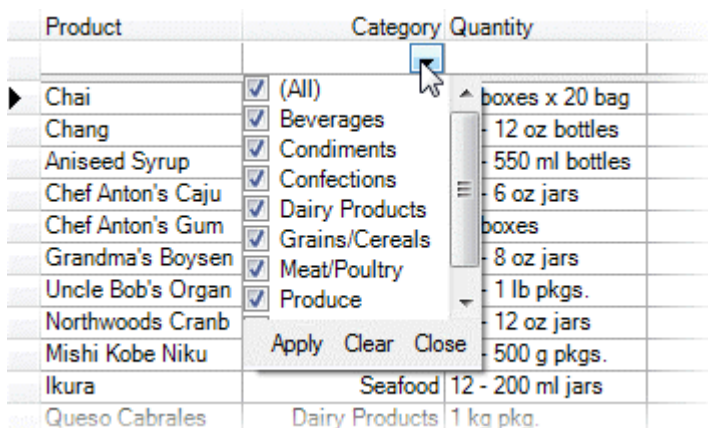
```
' Display the filter bar.
Me.ClTrueDBGrid1.FilterBar = True
' Allow the first column to be filtered by multiple items.
Me.ClTrueDBGrid1.Columns(1).FilterDropDown = True
' Allow the first column to be filtered by multiple items.
Me.ClTrueDBGrid1.Columns(1).FilterMultiSelect = True
```

To write code in C#

C#

```
// Display the filter bar.
this.clTrueDBGrid1.FilterBar = true;
// Allow the first column to be filtered by multiple items.
this.clTrueDBGrid1.Columns[1].FilterDropDown = true;
// Allow the first column to be filtered by multiple items.
this.clTrueDBGrid1.Columns[1].FilterMultiSelect = true;
```

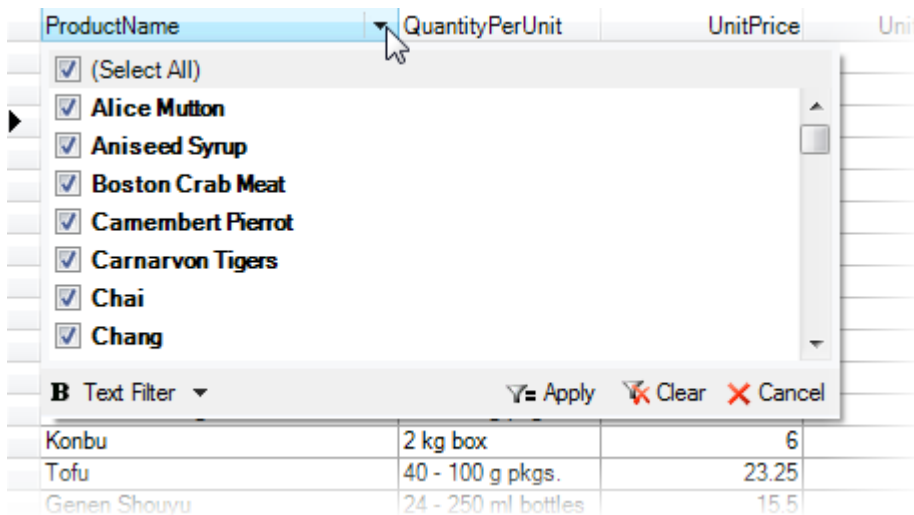
If you run the application, you'll notice that you can select the filter drop-down list to choose what to filter the column by:



Click check boxes to choose items that will be displayed. Checked items will be displayed and items with cleared check boxes will not be displayed. Click the **Apply** button in the filter bar to apply the filter criteria. Click the **Clear** button to clear the filter. Click the **Close** button to close the drop-down list.

Condition Filtering

True DBGrid for WinForms now includes flexible conditional filtering similar to the **FlexGrid for WinForms** style of filtering.



If you set the **FilterBar** property to **False**, and the **FilterDropDown** property to **True**, the **C1TrueDBGrid** control will allow using the new filters in the column.

To write code in Visual Basic

Visual Basic

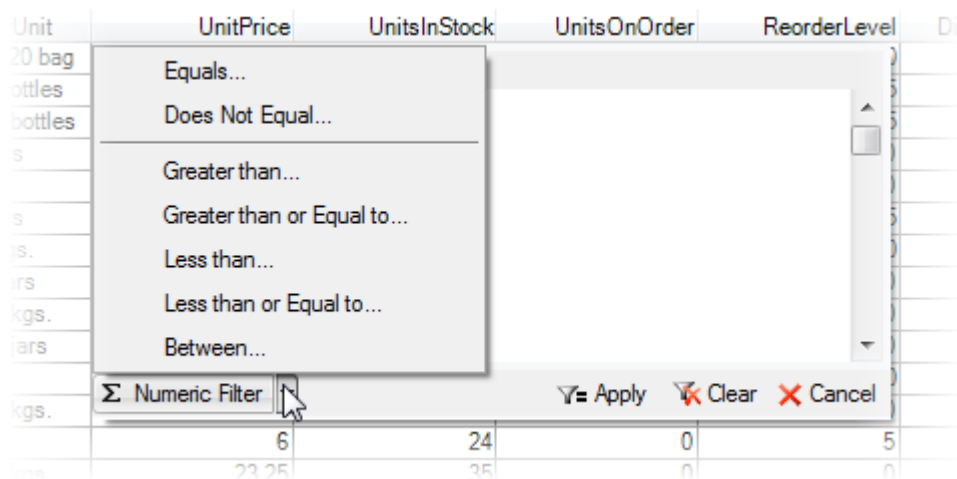
```
' Do not display the filter bar.
Me.C1TrueDBGrid1.FilterBar = False
' Allow the first column to be filtered by multiple items.
Me.C1TrueDBGrid1.Columns(1).FilterDropDown = True
```

To write code in C#

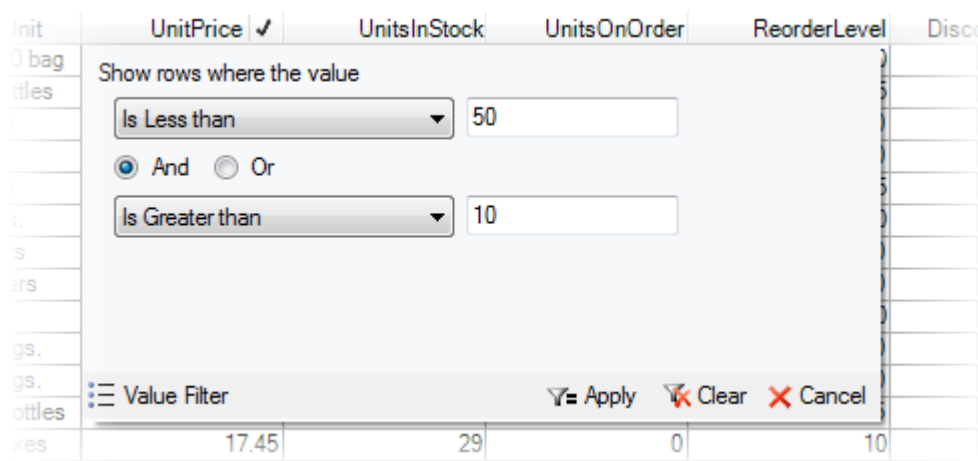
C#

```
// Do not display the filter bar.
this.c1TrueDBGrid1.FilterBar = false;
// Allow the first column to be filtered by multiple items.
this.c1TrueDBGrid1.Columns[1].FilterDropDown = true;
```

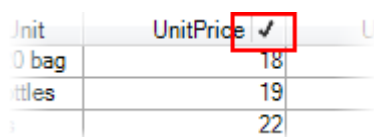
The data filtering feature follows the pattern used by **C1FlexGrid**. When users move the mouse over column headers, the grid displays a filter icon on the header. Clicking the icon invokes the filter editor which selects the data that should be displayed. Every column has a filter, and rows must pass all filters in order to be displayed.



The built-in filters include the value filter, which allows users to select specific values for display, as well as a condition filter, which allows them to specify up to two conditions using operators such as "greater than", "starts with", or "contains". This type of filter is recommended for filtering columns that contain "continuous" values such as numeric or date/time values.



Columns that have filters applied display the filter icon on their headers even when the mouse is not over them.



Custom Filtering

You can create your own filtering method using the [FilterDefinition](#) property to save/load custom filters in code.

You can apply one of a few pre-defined filters by reading the [FilterDefinition](#) property from a xml file like the following:

To write code in Visual Basic

Visual Basic

```
Private Sub ReadFilter(name As String)
    c1TrueDBGrid1.Splits(0).FilterDefinition = System.IO.File.ReadAllText(name &
```

```
".xml")  
End Sub
```

To write code in C#

```
C#  
  
void ReadFilter(string name)  
{  
    c1TrueDBGrid1.Splits[0].FilterDefinition = System.IO.File.ReadAllText(name +  
    ".xml");  
}
```

The custom filter can then be applied to the grid and saved as custom like the following:

To write code in Visual Basic

```
Visual Basic  
  
Private Sub SaveCustomFilter()  
    System.IO.File.WriteAllText("custom.xml",  
    Me.c1TrueDBGrid1.Splits(0).FilterDefinition)  
End Sub
```

To write code in C#

```
C#  
  
void SaveCustomFilter()  
{  
    System.IO.File.WriteAllText("custom.xml",  
    this.c1TrueDBGrid1.Splits[0].FilterDefinition);  
}
```



Note: For a complete sample using this FilterDefinition property, see the **FilterDefinitionTdbg** sample installed with **Winforms Edition**.

How to Use Splits

In **True DBGrid for WinForms**, a *split* is similar to the split window features of products such as Microsoft Excel and Word. Use splits to present data in multiple horizontal or vertical panes. These panes, or splits, can display data in different colors and fonts. The splits can scroll as a unit or individually and can display different sets of columns or the same set. Also use splits to prevent one or more columns or a set of rows from scrolling. Unlike other grid products, fixed (nonscrolling) columns or rows in **True DBGrid for WinForms** do not have to be at the left edge of the grid, but can be at the right edge or anywhere in the middle. Multiple groups of fixed columns or rows can exist within a grid. Splits open up an endless variety of possibilities for presenting data to users of your applications.

Whenever you use **True DBGrid for WinForms**, you are always using a split. A grid always contains at least one horizontal split, and the default values for the split properties are set so splits can be ignored until needed. Therefore, skip this chapter if you do not need to create and manipulate more than one split within a grid.

Create and manipulate splits by working with [Split](#) objects and the [SplitCollection](#) object. Since an individual column can be visible in one split but hidden in another, each [Split](#) object maintains its own collection of columns, known as [C1DisplayColumnCollection](#). This collection provides complete control over the appearance of each split and the columns it contains.

Referencing Splits and their Properties

A [C1TrueDBGrid](#) object initially contains a single horizontal split. If additional splits are created, you can determine or set the current split (that is, the split that has received focus) using the grid's [SplitIndex](#) property:

To write code in Visual Basic

Visual Basic

```
' Read the zero-based index of the current split.
Dim idx as Integer = Me.C1TrueDBGrid1.SplitIndex

' Set focus to the split with an index equal to Variable%.
Me.C1TrueDBGrid1.SplitIndex = idx
```

To write code in C#

C#

```
// Read the zero-based index of the current split.
int idx = this.c1TrueDBGrid1.SplitIndex;

// Set focus to the split with an index equal to Variable%.
this.c1TrueDBGrid1.SplitIndex = idx;
```

Each split in a grid is a different view of the same data source, and behaves just like an independent grid. If additional splits are created without customizing any of the split properties, all splits will be identical and each will behave very much like the original grid with one split.

Note that some properties, such as [RecordSelectors](#) and [MarqueeStyle](#), are supported by both the [C1TrueDBGrid](#) and [Split](#) objects. Three rules of thumb apply to properties that are common to a grid and its splits:

1. When you set or get the property of a [Split](#) object, you are accessing a specific split, and other splits in the same grid are not affected.
2. When you get the property of a [C1TrueDBGrid](#) object, you are accessing the same property within the current split.

3. When you set the property of a [C1TrueDBGrid](#) object, you are setting the same property within **all** splits.

To understand how these rules work in code, consider a grid with two horizontal splits, and assume that the current split index is 1. To determine which marquee style is in use, the following statements are equivalent:

To write code in Visual Basic

Visual Basic

```
marquee = Me.C1TrueDBGrid1.MarqueeStyle
marquee = Me.C1TrueDBGrid1.Splits(1).MarqueeStyle
marquee = Me.C1TrueDBGrid1.Splits(Me.C1TrueDBGrid1.SplitIndex).MarqueeStyle
```

To write code in C#

C#

```
marquee = this.c1TrueDBGrid1.MarqueeStyle;
marquee = this.c1TrueDBGrid1.Splits[1].MarqueeStyle;
marquee = this.c1TrueDBGrid1.Splits[this.csss1TrueDBGrid1.SplitIndex].MarqueeStyle;
```

To change the marquee style to a solid cell border for all of the splits in the grid, use:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.MarqueeStyle = MarqueeEnum.SolidCellBorder
```

To write code in C#

C#

```
this.c1TrueDBGrid1.MarqueeStyle = MarqueeEnum.SolidCellBorder;
```

Note that this statement is equivalent to:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).MarqueeStyle = MarqueeEnum.SolidCellBorder
Me.C1TrueDBGrid1.Splits(1).MarqueeStyle = MarqueeEnum.SolidCellBorder
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits(0).MarqueeStyle = MarqueeEnum.SolidCellBorder;
this.c1TrueDBGrid1.Splits(1).MarqueeStyle = MarqueeEnum.SolidCellBorder;
```

Likewise, to set the marquee style of each split to a different value:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).MarqueeStyle = MarqueeEnum.NoMarquee
Me.C1TrueDBGrid1.Splits(1).MarqueeStyle = MarqueeEnum.FloatingEditor
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits(0).MarqueeStyle = MarqueeEnum.NoMarquee;  
this.c1TrueDBGrid1.Splits(1).MarqueeStyle = MarqueeEnum.FloatingEditor;
```

These rules apply only to a [C1TrueDBGrid](#) object and its associated [Split](#) objects. No other object pairs possess similar relationships.

Split Properties Common to C1TrueDBGrid

The following properties, which are supported by both [Split](#) and [C1TrueDBGrid](#) objects, adhere to the rules described in the preceding section:

| Property | Description |
|-------------------------------------|---|
| AllowColMove | Enables interactive column movement. |
| AllowColSelect | Enables interactive column selection. |
| AllowRowSelect | Enables interactive row selection. |
| AllowRowSizing | Enables interactive row resizing. |
| AlternatingRowStyle | Controls whether even/odd row styles are applied to a split. |
| CaptionStyle | Controls the caption style for a split. |
| CurrentCellVisible | Sets/returns modification status of the current cell. |
| ExtendRightColumn | Sets/returns extended right column for a split. |
| FetchRowStyles | Controls whether the FetchRowStyle event will be fired. |
| FirstRow | Bookmark of row occupying first display line. |
| LeftCol | Returns the leftmost visible column. |
| MarqueeStyle | Sets/returns marquee style for a split. |
| RecordSelectors | Shows/hides selection panel at left border. |



Note: The [Caption](#) property is not included in this list, even though it is supported by both objects. Since grids and splits maintain separate caption bars, setting the **Caption** property of the grid does not apply the same string to each split caption.

Split-Only Properties Not Supported by C1TrueDBGrid

The following properties are supported by [Split](#) objects but not by [C1TrueDBGrid](#). Therefore, to apply a value to the entire grid, set the value for each split individually.

| Property | Description |
|---------------------------------------|---|
| AllowFocus | Allows cells within a split to receive focus. |
| HorizontalScrollGroup | Controls the horizontal scroll bar. |

| | |
|---------------------|---|
| Locked | True if data entry is prohibited for a split. |
| SplitSize | Sets/returns split width according to SizeMode. |
| SplitSizeMode | Controls whether a split is scalable or fixed size. |
| VerticalScrollGroup | Controls the vertical scroll bar. |

Split Matrix Notation

When the grid contains both horizontal and vertical splits, it is said to be organized in a two-dimensional split matrix. Reference and access to the properties of the split objects in the matrix is accomplished with a two-dimensional matrix notation. The index for a particular split in a split matrix is the split row, then the column delimited by a comma. For instance, accessing the second vertical split (column) in the third horizontal split (row) would look like the following:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits.Item(2,1).Style.ForeColor = System.Drawing.Color.Blue
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[2,1].Style.ForeColor = System.Drawing.Color.Blue;
```



Note: Notice that the **Item** property is used in the previous example. When accessing a split through split matrix notation, the **Item** property must be explicitly specified. When accessing a split in a grid with a one-dimensional structure, the **Item** property is taken as implicit and can be omitted.

For instance, accessing the second split in a grid with only horizontal splits would look like the following:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(1).Style.ForeColor = System.Drawing.Color.Blue
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits(1).Style.ForeColor = System.Drawing.Color.Blue;
```

Creating and Removing Splits

In code, you must create and remove splits using the [RemoveHorizontalSplit](#), [InsertHorizontalSplit](#), [RemoveVerticalSplit](#), and [RemoveHorizontalSplit](#) methods. Each method takes a zero-based split index:

To write code in Visual Basic

Visual Basic

```
Dim S As C1TrueDBGrid.Split
```

```
' Create a split with index 7.
Me.C1TrueDBGrid1.InsertVerticalSplit(7)

' Remove the split with index 5.
Me.C1TrueDBGrid1.RemoveVerticalSplit(5)
```

To write code in C#

```
C#

C1TrueDBGrid.Split S;

// Create a split with index 7.
this.c1TrueDBGrid1.InsertVerticalSplit(7);

// Remove the split with index 5.
this.c1TrueDBGrid1.RemoveVerticalSplit(5);
```

You can determine the number of splits in a grid using the [SplitCollection Count](#) property:

To write code in Visual Basic

```
Visual Basic

' Set variable equal to the number of splits in C1TrueDBGrid1.
variable = Me.C1TrueDBGrid1.Splits.Count
```

To write code in C#

```
C#

// Set variable equal to the number of splits in C1TrueDBGrid1.
variable = this.c1TrueDBGrid1.Splits.Count;
```

You can iterate through all splits using the **Count** property, for example:

To write code in Visual Basic

```
Visual Basic

For n = 0 To Me.C1TrueDBGrid1.Splits.Count - 1
    Debug.WriteLine (Me.C1TrueDBGrid1.Splits(n).Caption)
Next n
```

To write code in C#

```
C#

for (n = 0 ; n < this.c1TrueDBGrid1.Splits.Count; n++)
{
    Console.WriteLine (this.c1TrueDBGrid1.Splits[n].Caption);
}
```

Of course, a more efficient way to code this would be to use a For Each...Next loop:

To write code in Visual Basic

Visual Basic

```
Dim S As C1TrueDBGrid.Split
For Each S In Me.C1TrueDBGrid1.Splits
    Debug.WriteLine (S.Caption)
Next
```

To write code in C#

C#

```
C1TrueDBGrid.Split S;
foreach (S In this.c1TrueDBGrid1.Splits)
{
    Console.WriteLine (S);
}
```

The new [Split](#) object will inherit all of its properties from the last object in the collection.

Working with Columns in Splits

Each split in a **True DBGrid for WinForms** control maintains its own collection of columns. The [C1DisplayColumnCollection](#) object provides access to both split-specific display properties for columns inside a split. The split-specific properties of the [C1DisplayColumnCollection](#) allow for tremendous flexibility in controlling the look and behavior of individual splits. The grid is connected to a single data source, so the splits just present different views of the same data. Therefore, the [C1DisplayColumnCollection](#) in each split contains the same number of columns and the columns are bound to the same data fields.

However, the values of other [C1DisplayColumn](#) object properties, such as [Visible](#), may vary from split to split. These properties are said to be *split-specific*. For example, a column created in code is not visible by default. Thus, the *LastName* column created in the preceding example is invisible in all splits. The following code makes it visible in the second split:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(1).DisplayColumns("LastName").Visible = True
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits(1).DisplayColumns("LastName").Visible = true;
```

Since [Visible](#) is a split-specific property, the *LastName* column remains invisible in other splits.

Sizing and Scaling Splits

True DBGrid for WinForms provides full control over the size and scaling of individual splits. Configure a split to occupy an exact width or height, hold a fixed number of columns or rows, or adjust its size proportionally in relation to other splits. When initially starting out with **True DBGrid for WinForms** controls, splits can still be used in a variety

of ways without having to master all of the details.

At run time, the actual size of a [Split](#) object depends upon its [SplitSize](#) and [SplitSizeMode](#) properties. The [SplitSizeMode](#) property specifies the unit of measurement; the [SplitSize](#) property specifies the number of units. **TrueDBGrid for WinForms** supports three different sizing modes for splits, as determined by the setting of the [SplitSizeMode](#) property:

| Mode | Description |
|------------------------------|---|
| SizeModeEnum.Scalable | Denotes relative width in relation to other splits. |
| SizeModeEnum.Exact | Specifies a fixed width in container coordinates. |
| SizeModeEnum.NumberofColumns | Specifies a fixed number of columns. |

A scalable split uses the value of its [SplitSize](#) property to determine the percentage of space the split will occupy. For any scalable split, the percentage is determined by dividing its [SplitSize](#) value by the sum of the [SplitSize](#) values of all other scalable splits. Thus, consider the [SplitSize](#) property of a scalable split to be the numerator of a fraction, the denominator of which is the sum of the scalable split sizes. Scalable splits compete for the space remaining after nonscalable splits have been allocated. By default, all splits are scalable, so they compete for the entire grid display region. [SplitSizeMode](#) is always **Scalable** when a grid contains only one split.

An exact split uses the value of its [SplitSize](#) property as its fixed width in container coordinates. Exact splits will be truncated if they will not fit within the horizontal grid boundaries. This mode is not applicable when a grid contains only one split.

A fixed-column split uses the [SplitSize](#) property to indicate the exact number of columns that should always be displayed within the split. These splits automatically reconfigure the entire grid if the size of the displayed columns changes (either by code or user interaction), or if columns in the split are scrolled horizontally so that the widths of the displayed columns are different. This mode is primarily used to create fixed columns that do not scroll horizontally. However, it can be used for a variety of other purposes as well. This mode is not applicable when a grid contains only one split.

Note that when there is only one split (the grid's default behavior), the split spans the entire width of the grid, the [SplitSizeMode](#) property is always **Scalable**, and the [SplitSize](#) property is always **1**. Setting either of these properties has no effect when there is only one split. If there are multiple splits, and then remove all but one so the [SplitSizeMode](#) and [SplitSize](#) properties of the remaining split automatically revert to 0 and 1, respectively.

By default, the [SplitSizeMode](#) property for a newly created split is [SizeModeEnum.Scalable](#), and the [SplitSize](#) property is set to 1. For example, two additional splits can be created with the following code:

To write code in Visual Basic

Visual Basic

```
' Create a Split on the left.
Me.C1TrueDBGrid1.InsertHorizontalSplit(0)

' Create another.
Me.C1TrueDBGrid1.InsertHorizontalSplit(0)
```

To write code in C#

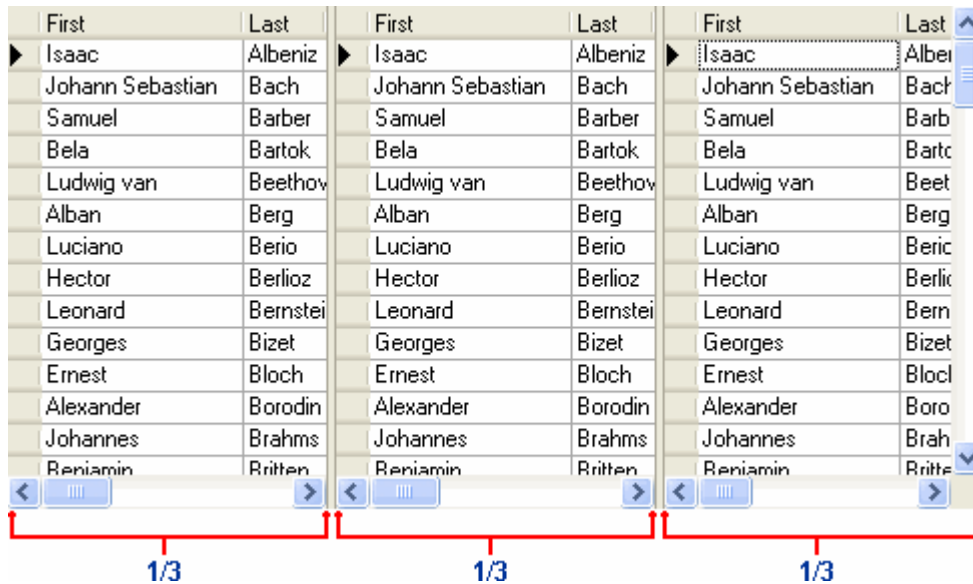
C#

```
// Create a Split on the left.
this.c1TrueDBGrid1.InsertHorizontalSplit(0);
```

```
// Create another.
this.clTrueDBGrid1.InsertHorizontalSplit(0);
```

The resulting grid display will appear as follows:

The resulting grid display will appear as follows:



Notice that each split occupies 1/3 of the total grid space. This is because there are three scalable splits, and each split has a SplitSize of 1. If the sizes of the splits are changed to 1, 2, and 3, respectively:

To write code in Visual Basic

Visual Basic

```
' Change relative size to 1.
Me.clTrueDBGrid1.Splits(0).SplitSize = 1

' Change relative size to 2.
Me.clTrueDBGrid1.Splits(1).SplitSize = 2

' Change relative size to 3.
Me.clTrueDBGrid1.Splits(2).SplitSize = 3
```

To write code in C#

C#

```
// Change relative size to 1.
this.clTrueDBGrid1.Splits[0].SplitSize = 1;

// Change relative size to 2.
this.clTrueDBGrid1.Splits[1].SplitSize = 2;

// Change relative size to 3.
this.clTrueDBGrid1.Splits[2].SplitSize = 3;
```

The resulting grid display will appear as follows:

| First | Last | First | Last |
|-------------|-----------|------------------|-----------|
| Isaac | Albeniz | Isaac | Albeniz |
| Johann Seba | Bach | Johann Sebastian | Bach |
| Samuel | Barber | Samuel | Barber |
| Bela | Bartok | Bela | Bartok |
| Ludwig van | Beethov | Ludwig van | Beethoven |
| Alban | Berg | Alban | Berg |
| Luciano | Berio | Luciano | Berio |
| Hector | Berlioz | Hector | Berlioz |
| Leonard | Bernstein | Leonard | Bernstein |
| Georges | Bizet | Georges | Bizet |
| Ernest | Bloch | Ernest | Bloch |
| Alexander | Borodin | Alexander | Borodin |
| Johannes | Brahms | Johannes | Brahms |
| Benjamin | Britten | Benjamin | Britten |

Diagram showing the split sizes: 1/6, 1/3, and 1/2.

Notice the sum of the split sizes (1+2+3) is 6, so the size of each split is a fraction with the numerator being the value of its SplitSize property and a denominator of 6.

When a split's SplitSizeMode is set to [SizeModeEnum.Exact](#), that split receives space before the other splits. This behavior is somewhat more complex, but understanding how scalable splits work is helpful. For example, assume that splits are set in the following way:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).SplitSizeMode = C1.Win.C1TrueDBGrid.SizeModeEnum.Scalable
Me.C1TrueDBGrid1.Splits(0).SplitSize = 1

Me.C1TrueDBGrid1.Splits(1).SplitSizeMode = C1.Win.C1TrueDBGrid.SizeModeEnum.Exact
Me.C1TrueDBGrid1.Splits(1).SplitSize = 250

Me.C1TrueDBGrid1.Splits(2).SplitSizeMode = C1.Win.C1TrueDBGrid.SizeModeEnum.Scalable
Me.C1TrueDBGrid1.Splits(2).SplitSize = 2
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].SplitSizeMode =
C1.Win.C1TrueDBGrid.SizeModeEnum.Scalable;
this.c1TrueDBGrid1.Splits[0].SplitSize = 1;

this.c1TrueDBGrid1.Splits[1].SplitSizeMode = C1.Win.C1TrueDBGrid.SizeModeEnum.Exact;
this.c1TrueDBGrid1.Splits[1].SplitSize = 250;

this.c1TrueDBGrid1.Splits[2].SplitSizeMode =
C1.Win.C1TrueDBGrid.SizeModeEnum.Scalable;
this.c1TrueDBGrid1.Splits[2].SplitSize = 2;
```

After configuring the splits in this way, the resulting grid display will look like this.

| First | First | Last | Count | First | Last |
|------------|------------------|-----------|---------|------------------|------|
| Isaac | Isaac | Albeniz | Spain | Isaac | Albe |
| Johann Seb | Johann Sebastian | Bach | Germa | Johann Sebastian | Bac |
| Samuel | Samuel | Barber | United | Samuel | Barl |
| Bela | Bela | Bartok | Hunga | Bela | Barl |
| Ludwig van | Ludwig van | Beethoven | Germa | Ludwig van | Bee |
| Alban | Alban | Berg | Austria | Alban | Berl |
| Luciano | Luciano | Berio | Italy | Luciano | Beri |
| Hector | Hector | Berlioz | France | Hector | Berl |
| Leonard | Leonard | Bernstein | United | Leonard | Berl |
| Georges | Georges | Bizet | France | Georges | Bize |
| Ernest | Ernest | Bloch | Switze | Ernest | Bloc |
| Alexander | Alexander | Borodin | Russia | Alexander | Borl |
| Johannes | Johannes | Brahms | Germa | Johannes | Bral |
| Benjamin | Benjamin | Britten | Engla | Benjamin | Brit |

1/3 250 Pixels 2/3

The fixed-size split in the middle (Split1) is configured to exactly 250 pixels, and the remaining splits compete for the space remaining in the grid. Since the remaining splits are both scalable splits, they divide the remaining space among themselves according to the percentages calculated using their SplitSize property values. So, the leftmost split occupies 1/3 of the **remaining** space, and the rightmost split occupies 2/3.

Splits with SplitSizeMode set to [SizeModeEnum.NumberOfColumns](#) behave almost identically to exact splits, except their size is determined by the width of an integral number of columns. The width, however, is dynamic, so resizing the columns or scrolling so that different columns are in view will cause the entire grid to reconfigure itself.

Avoid creating a grid with no scalable splits. Although **True DBGrid for WinForms** handles this situation, it is difficult to work with a grid configured in this way. For example, if no splits are scalable, all splits will have an exact size, which may not fill the entire horizontal width of the grid. If the total width of the splits is too short, **True DBGrid for WinForms** displays a "null-zone" where there are no splits. If the total width of the splits is wider than the grid, then **True DBGrid for WinForms** will show only the separator lines for the splits that cannot be shown.

Creating and Resizing Splits through User Interaction

Splits can be created and resized in code. However, users can also create and resize splits interactively by setting the [AllowHorizontalSplit](#) or [AllowVerticalSplit](#) property of a grid to **True**. By default, both properties are set to **False**, preventing users from creating and resizing splits.

A typical grid with these properties set to **False** is shown in the following figure. Notice that there is no split box at the left edge of the horizontal scroll bar or at the top of the vertical scroll bar.

| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |
| Luciano | Berio | Italy |
| Hector | Berlioz | France |

The new split will inherit its properties from the original split. The [SplitSizeMode](#) properties of both splits will be automatically set to [SizeModeEnum.Scalable](#), regardless of the SplitSizeMode of the original split. The SplitSize properties of both splits will be set to the correct ratio of the splits' sizes. The values of the [SplitSize](#) properties may end up being rather large. This is because **True DBGrid for WinForms** needs to choose the least common denominator for the total split size, and the user may drag the pointer to an arbitrary position.

Vertical Splits

If the split's AllowVerticalSplit property is set to **True**:

To write code in Visual Basic

Visual Basic

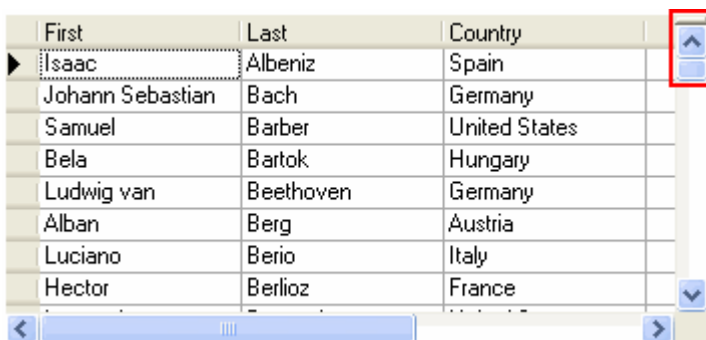
```
Me.C1TrueDBGrid1.AllowVerticalSplit = True
```

To write code in C#

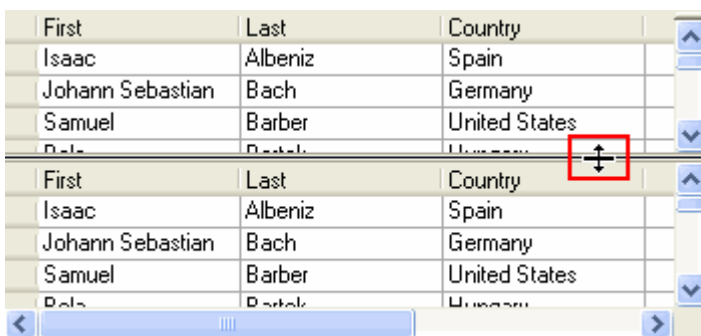
C#

```
this.c1TrueDBGrid1.AllowVerticalSplit = true;
```

A split box will appear at the top edge of the vertical scroll bar, and when the cursor is over the split box, it will turn into a double horizontal bar with vertical arrows. Dragging the cursor down from the split box creates a new split.



Once a split has been created, dragging the cursor up or down adjusts the relative size of the splits.



Horizontal Splits

If the split's AllowHorizontalSplit property is set to **True**:

To write code in Visual Basic

Visual Basic

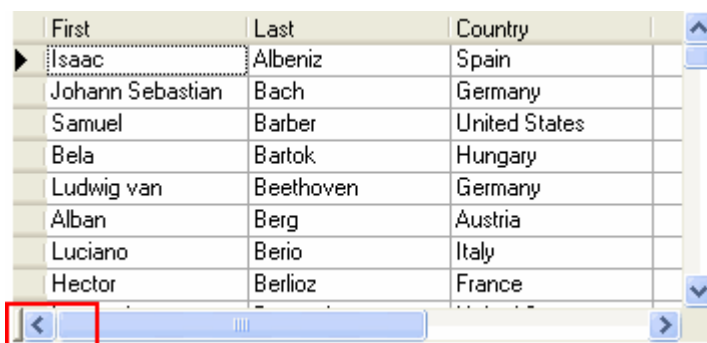
```
Me.ClTrueDBGrid1.AllowHorizontalSplit = True
```

To write code in C#

C#

```
this.clTrueDBGrid1.AllowHorizontalSplit = true;
```

A split box will appear at the left edge of the horizontal scroll bar, and when the cursor is over the split box, it will turn into a double vertical bar with horizontal arrows. Dragging the cursor to the left from the split box creates a new split.



Once a split has been created, dragging the cursor to the left or right adjusts the relative size of the splits.



Summary

Splits can always be created or resized in code, but the `AllowHorizontalSplit` and `AllowVerticalSplit` property control whether users can create or resize splits interactively at run time.

The user can resize the relative sizes of two splits only if both splits' `AllowSizing` properties are **True**. When the user completes a resize operation, the total size of the two splits remains unchanged, but the `SplitSizeMode` properties of both splits will automatically be set to `SizeModeEnum.Scalable` regardless of their previous settings. The `SplitSize` properties of the two splits will be set to reflect the ratio of their new sizes.

Vertical Scrolling and Split Groups

By default, the grid has only one horizontal split, with split index 0, and its `HScrollBar` and `VScrollBar` style property is set to `ScrollBarStyleEnumAutomatic`. That is, the horizontal or vertical scroll bar will appear as necessary depending

upon the column widths and the number of data rows available. The default split's [HorizontalScrollGroup](#) and [VerticalScrollGroup](#) properties are set to 1. Splits having the same scrollgroup property setting will scroll vertically or horizontally together. When a new split is created, it will inherit both the state of the scroll bars and the Scroll Group properties from the parent split. If all of the splits belonging to the same [HorizontalScrollGroup](#) or [VerticalScrollGroup](#) have their [HScrollBar](#) and [VScrollBar](#) style property is set to [ScrollBarStyleEnumAutomatic](#), then **True DBGrid for WinForms** will display the vertical scroll bar or horizontal scroll bar only at the rightmost or bottommost split of the scroll group. Manipulating this single scroll bar will cause all splits in the same scroll group to scroll simultaneously.

For example, two additional splits can be created with the following code:

To write code in Visual Basic

Visual Basic

```
' Create a Split at the left.
Me.C1TrueDBGrid1.InsertHorizontalSplit(0)

' Create another.
Me.C1TrueDBGrid1.InsertHorizontalSplit(0)
```

To write code in C#

C#

```
// Create a Split at the left.
this.c1TrueDBGrid1.InsertHorizontalSplit(0);

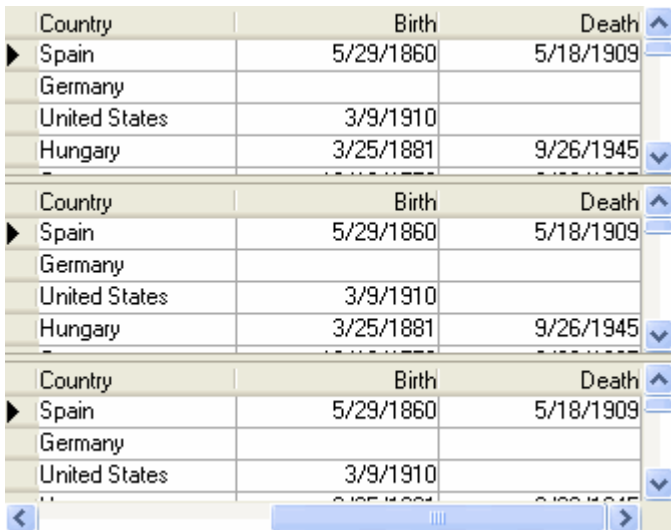
// Create another.
this.c1TrueDBGrid1.InsertHorizontalSplit(0);
```

The resulting grid display will display as follows:



All three splits will have the same [HScrollBar](#) and [VScrollBar](#) settings and [VerticalScrollGroup](#) setting of 1. However, only one vertical scroll bar will be displayed, within the rightmost split. When the user operates this scroll bar, all three splits will scroll simultaneously.

Vertical splits react in the same manner. After adding two vertical splits to the grid, all of the splits have the same [HorizontalScrollGroup](#) value of 1. Thus there is only one horizontal scroll bar at the bottom of the grid, and if this scroll bar is scrolled all three splits will scroll simultaneously.



| Country | Birth | Death |
|---------------|-----------|-----------|
| Spain | 5/29/1860 | 5/18/1909 |
| Germany | | |
| United States | 3/9/1910 | |
| Hungary | 3/25/1881 | 9/26/1945 |

Change one of the scroll group properties of the splits to create split groups that scroll independently. In the preceding example, setting the [HorizontalScrollGroup](#) property of the middle split to **2** creates a new scroll group:

To write code in Visual Basic

Visual Basic

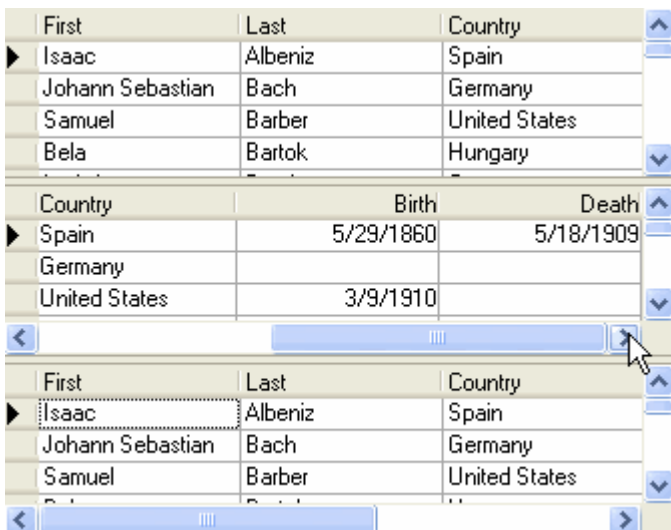
```
Me.c1TrueDBGrid1.Splits.Item(0,1).HorizontalScrollGroup = 2
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0,1].HorizontalScrollGroup = 2;
```

After this statement executes, scrolling the middle split will not disturb the others, as shown in the following figure.



| First | Last | Country |
|------------------|---------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |

Note that the middle split now contains a horizontal scroll bar. This scroll bar operates only on the middle split, since it is the only one with its [HorizontalScrollGroup](#) property equal to **2**. The horizontal scroll bar in the bottommost split now controls the bottom and top splits only. It no longer affects the middle split.

A common application of this feature is to create two independent split groups so that users can compare field values from different records by scrolling each split to view a different set of rows.

Horizontal Scrolling and Fixed Columns

Scrolling is independent for each split. Often, one or more columns need to be prevented from scrolling horizontally or vertically so that the columns will always be in view. **True DBGrid for WinForms** provides an easy way to keep any number of columns from scrolling at any location within the grid (even in the middle!) by setting a few split properties.

As an example, with a grid with three horizontal splits, the following code will "fix" columns 0 and 1 in the middle split:

To write code in Visual Basic

Visual Basic

```
' Hide all columns in Splits(1) except for columns 0 and 1.
Dim Cols As C1TrueDBGrid.C1DisplayColumnCollection
Dim C As C1TrueDBGrid.C1DisplayColumn

Cols = Me.C1TrueDBGrid1.Splits(1).DisplayColumns
For Each C In Cols
    C.Visible = False
Next C
Cols(0).Visible = True
Cols(1).Visible = True

' Configure Splits(1) to display exactly two columns, and disable resizing.
With Me.C1TrueDBGrid1.Splits(1)
    .SplitSizeMode = SizeModeEnum.NumberOfColumns
    .SplitSize = 2
    .AllowHorizontalSizing = False
End With
```

To write code in C#

C#

```
// Hide all columns in Splits[1] except for columns 0 and 1.
C1TrueDBGrid.C1DisplayColumnCollection Cols;
C1TrueDBGrid.C1DisplayColumn C;

Cols = this.c1TrueDBGrid1.Splits[1].DisplayColumns
foreach (C In Cols)
{
    C.Visible = false;
}
Cols(0).Visible = true;
Cols(1).Visible = true;

// Configure Splits[1] to display exactly two columns, and disable resizing.
this.c1TrueDBGrid1.Splits[1].SplitSizeMode = SizeModeEnum.NumberOfColumns;
this.c1TrueDBGrid1.Splits[1].SplitSize = 2;
this.c1TrueDBGrid1.Splits[1].AllowHorizontalSizing = false;
```

Usually, if columns 0 and 1 are kept from scrolling in one split, it will be desirable to have them invisible in the other

splits:

To write code in Visual Basic

Visual Basic

```
' Make columns 0 and 1 invisible in splits 0 and 2.
Dim Cols As C1TrueDBGrid.C1DisplayColumnCollection
Cols = Me.C1TrueDBGrid1.Splits(0).DisplayColumns
Cols(0).Visible = False
Cols(1).Visible = False
Cols = Me.C1TrueDBGrid1.Splits(2).DisplayColumns
Cols(0).Visible = False
Cols(1).Visible = False
```

To write code in C#

C#

```
// Make columns 0 and 1 invisible in splits 0 and 2.
C1TrueDBGrid.C1DisplayColumnCollection Cols;
Cols = this.c1TrueDBGrid1.Splits[0].DisplayColumns;
Cols[0].Visible = false;
Cols[1].Visible = false;
Cols = this.c1TrueDBGrid1.Splits[2].DisplayColumns;
Cols[0].Visible = false;
Cols[1].Visible = false;
```

Navigation Across Splits

Navigation across splits is controlled by the grid's [TabAcrossSplits](#) property and each split's [AllowFocus](#) property. Navigation across splits is best discussed with grid navigation as a whole. For more information, please refer to [Run-Time Interaction](#).

How to Use Styles

True DBGrid for WinForms uses a style model similar to that of Microsoft Word and Excel to simplify the task of customizing a grid's appearance. A [Style](#) object is a combination of font, color, picture, and formatting information comprising the following properties:

| Property | Description |
|---|---|
| Alpha | Gets or sets the alpha component when the style is rendered. |
| BackColor | Gets or sets the background color associated with a Style. |
| BackColor2 | Gets or sets the background color associated with a Style. |
| BackgroundImage | Gets or sets the background image associated with a Style. |
| BackgroundPictureDrawMode | Gets or sets the rendering method for a BackgroundImage. |
| Borders | Gets the GridBorders associated with this Style. |
| Font | Gets or sets the Font associated with a Style. |
| ForeColor | Gets or sets the foreground color associated with a Style. |
| ForegroundImage | Gets or sets the foreground image associated with a style. |
| ForegroundPicturePosition | Gets or sets the position that the ForegroundImage is rendered. |
| GammaCorrection | Gets or sets a value indicating whether gamma correction is enabled when a linear gradient style is rendered. |
| GradientMode | Specifies the direction of a linear gradient. |
| HorizontalAlignment | Gets or sets the horizontal text alignment. |
| Locked | Gets or sets a value indicating whether data entry is permitted for the associated object. |
| Name | Gets or sets the name of the Style. |
| Padding | Gets or sets the spacing between cell content and its edges. |
| Trimming | Gets or sets the trim characters for a string that does not completely fit into a layout shape. |
| VerticalAlignment | Gets or sets the vertical text alignment. |
| WrapText | Gets or sets a value indicating whether text is word-wrapped when it does not fit into a layout shape. |

Built-In Named Styles

When a grid is first created, it has a collection of built-in named styles that control various aspects of its display. For example, the Heading style determines the attributes used to display column headers. At design time, change the appearance of the grid as a whole by modifying the built-in named styles in the **C1TrueDBGrid Styles Editor**. At run time, the [GridStyleCollection](#) provides access to the same set of named styles. Initially, all grids contain ten built-in styles, which control the display of the following grid elements:

| Element | Description |
|---------|-------------|
|---------|-------------|

| | |
|------------------------|--|
| Caption | Grid and split caption bars. |
| Editor | Cell editor within grid. |
| EvenRow | Data cells in even numbered rows. |
| Filter Bar | Data in the filter bar columns. |
| Footer | Column footers. |
| Group | Group columns in grid grouping area. |
| Heading | Column headers. |
| HighlightRow | Data cells in highlighted rows. |
| Inactive | Column headings when another column has focus. |
| Normal | Data cells in unselected, unhighlighted rows. |
| OddRow | Data cells in odd numbered rows. |
| Record Selector | Data in the record selector column. |
| Selected | Data cells in selected rows. |

A *selected row* is one whose bookmark has been added to the [SelectedRowCollection](#), either in code or through user interaction. The term *highlighted row* refers to the current row when the [MarqueeStyle](#) property is set to [MarqueeEnum.HighlightRow](#) or [MarqueeEnum.HighlightRowRaiseCell](#).

The EvenRow and OddRow styles are used only when the [AlternatingRows](#) property is set to **True**.

Named Style Defaults

As in Microsoft Word, a [Style](#) object in **True DBGrid** can inherit its characteristics from another style, referred to as the parent style. For a newly created grid, the Normal style is the parent (or grandparent) of all named styles. Its default properties are as follows:

| Property | Setting |
|---|---|
| Alpha | 255 |
| BackColor | System.Drawing.Color.White |
| BackColor2 | System.Drawing.Color.White |
| BackgroundImage | None |
| BackgroundPictureDrawMode | <i>BackgroundPictureDrawModeEnum.Stretch</i> |
| Font | Microsoft Sans Serif, 8.25pt |
| ForeColor | System.Drawing.Color.Black |
| ForegroundImage | None |
| ForegroundPicturePosition | <i>ForegroundPicturePositionEnum.LeftOfText</i> |
| GammaCorrection | False |
| GradientMode | None |
| HorizontalAlignment | <i>AlignHorzEnum.General</i> |

| | |
|-------------------|--------------------------|
| Locked | False |
| Padding | 0, 0, 0, 0 |
| Trimming | Character |
| VerticalAlignment | <i>AlignVertEnum.Top</i> |
| WrapText | False |

The Heading and Footing styles are defined similarly. Each inherits from the Normal style, and each overrides the following properties:

| Property | Setting |
|-------------------|-------------------------------------|
| BackColor | System.Drawing.SystemColors.Control |
| ForeColor | System.Drawing.Color.Black |
| VerticalAlignment | <i>AlignVertEnum.Center</i> |

The Heading style overrides one additional property that the Footing style does not:

| Property | Setting |
|----------|---------|
| WrapText | True |

The Selected style also inherits from Normal and overrides two color properties:

| Property | Setting |
|-----------|---|
| BackColor | System.Drawing.SystemColors.Highlight |
| ForeColor | System.Drawing.SystemColors.HighlightText |

The same is **True** of the HighlightRow style, which uses the inverse of the color settings for the default Normal style:

| Property | Setting |
|-----------|---|
| BackColor | System.Drawing.SystemColors.Text |
| ForeColor | System.Drawing.SystemColors.HighlightText |

The EvenRow, OddRow, and FilterBar styles inherit from Normal, but only the EvenRow style overrides any properties:

| Property | Setting |
|-----------|---------------------------|
| BackColor | System.Drawing.Color.Aqua |

The only styles that do not inherit directly from Normal are the Caption and RecordSelector styles, which inherit from the Heading style. The reason that grid and split captions are centered by default is that the Caption style specifies the following property:

| Property | Setting |
|---------------------|----------------------|
| HorizontalAlignment | AlignHorzEnum.Center |

Named Style Inheritance

To see how named style inheritance works, place a grid on a form and set the [Caption](#) property of the grid and its default columns. Set the [FooterText](#) property of the default columns and set the [ColumnFooters](#) property of the grid to **True**. The grid should look something like this.

| Caption | |
|----------|----------|
| Column 0 | Column 1 |
| * | |
| | |
| Footer 0 | Footer 1 |

In the **C1TrueDBGrid Style Editor**, select **Normal** from the left pane and expand the [Font](#) node. Set the **Bold** property to **True**. Note that the column headers, column footers, and grid caption are all bold, since all built-in styles inherit from the Normal style or one of its children.

| Caption | |
|----------|----------|
| Column 0 | Column 1 |
| * | |
| | |
| Footer 0 | Footer 1 |

Next, select **Heading** from the left pane, and in the right pane select the [ForeColor](#) property. Click the **Web** tab, and then select **Navy**. Note that the text color of both the column headers and the grid's caption bar is now white, since the Caption style inherits its color properties from the Heading style. The column footers remain the same because the Footer style inherits from the Normal style, not the Heading style.

| Caption | |
|----------|----------|
| Column 0 | Column 1 |
| * | |
| | |
| Footer 0 | Footer 1 |

Finally, select **Caption** from the left pane and in the right pane select its [BackColor](#) property. Click the **Web** tab, and then select **AliceBlue**. Note that the background color of the column headers is not changed, and that the Caption style continues to inherit its text color from its parent style, Heading.

| Caption | |
|----------|----------|
| Column 0 | Column 1 |
| * | |
| | |
| Footer 0 | Footer 1 |

Modifying Named Styles

Change the appearance of the overall grid at design time by using .NET's collection editors to modify the

[GridStyleCollection](#). For example, to force all column headers to center their caption text, change the [HorizontalAlignment](#) property of the built-in Heading style to [AlignHorzEnum.Center](#).

The following statement accomplishes the same result in code:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Styles("Heading").HorizontalAlignment = AlignHorzEnum.Center
```

To write code in C#

C#

```
this.C1TrueDBGrid1.Styles["Heading"].HorizontalAlignment = AlignHorzEnum.Center;
```

However, it is not necessary to use the **C1TrueDBGrid Style Editor** or manipulate named members of the [GridStyleCollection](#) in code, as the grid and its component objects expose several properties that return [Style](#) objects. As the next section describes, the appearance of the grid can be fine-tuned by manipulating these objects directly. For more information see [Using the C1TrueDBGrid Style Editor](#).

Working with Style Properties

Just as a document template in Microsoft Word specifies the overall appearance of individual paragraphs in a document, the named members of the [GridStyleCollection](#) object provide an overall display template for a [C1TrueDBGrid](#) or [C1TrueDBDropDown](#) control. However, to customize the appearance of individual [Split](#) or [C1DisplayColumn](#) objects, modify the appropriate [Style](#) object property:

| Property | Description |
|-------------------------------------|---|
| CaptionStyle | Controls the caption style for an object. |
| EditorStyle | Controls the editor style for an object. |
| EvenRowStyle | Controls the row style for even-numbered rows. |
| FilterBarStyle | Controls the style of the columns in the filter bar. |
| FooterStyle | Controls the footing style for an object. |
| HeadingStyle | Controls the heading style for an object. |
| HighlightRowStyle | Controls the marquee style when set to Highlight Row . |
| InactiveStyle | Controls the inactive heading style for an object. |
| OddRowStyle | Controls the row style for odd-numbered rows. |
| RecordSelectorStyle | Controls the record selector style for an object. |
| SelectedStyle | Controls the selected row/column style for an object. |
| Style | Controls the normal style for an object. |

Modifying a Style Property Directly

Customize the appearance of a grid component by modifying one or more members of an appropriate style property. For example, to make the grid's caption text bold, change the **Font** object associated with the [CaptionStyle](#) property.

At design time, this is done by expanding the **CaptionStyle** tree node on the Properties window, expanding the **Font** node, and setting the **Bold** property to **True**. The change is committed to the grid when you click out of this particular property.

Note when switching to the **C1TrueDBGrid Style Editor**, it will be seen that the built-in Caption style has not changed.

This means that the following statements are **not** equivalent:

To write code in Visual Basic

Visual Basic

```
Dim myfont As New Font(Me.C1TrueDBGrid1.Font, FontStyle.Bold)
Me.C1TrueDBGrid1.CaptionStyle.Font = myfont

Me.C1TrueDBGrid1.Styles("Caption").Font = myfont
```

To write code in C#

C#

```
Font myfont = new Font(this.c1TrueDBGrid1.Font, FontStyle.Bold);
this.c1TrueDBGrid1.CaptionStyle.Font = myfont;

this.c1TrueDBGrid1.Styles["Caption"].Font = myfont;
```

The first statement specifies the font of the grid's caption bar; because this is a root style, the named Caption style also changes.

Named Styles vs. Anonymous Styles

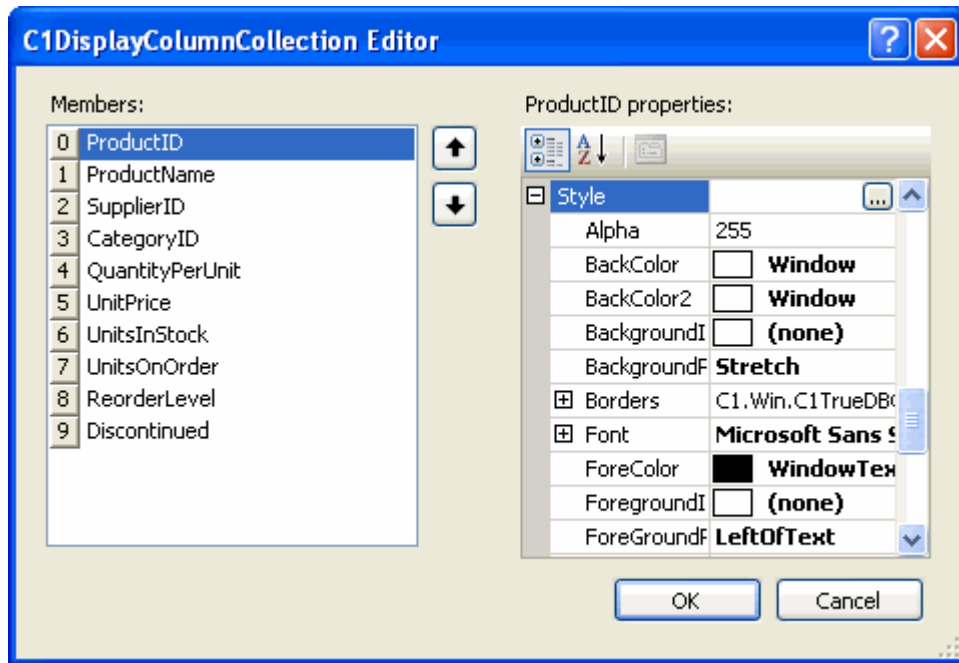
When setting style properties at design time, it is important to understand the distinction between named styles and the *anonymous styles* exposed by grid properties.

Named styles provide templates that govern the appearance of the grid, its splits, and its columns. At design time, create, modify, and delete named styles using the **GridStyleCollection Editor**. At run time, the [GridStyleCollection](#) is used to represent the same set of named [Style](#) objects.

Anonymous styles are **not** members of the [GridStyleCollection](#). However, anonymous styles are provided so that the appearance of an individual split or column can be easily and directly customized without having to define a separate named style.

The following analogy should help to clarify the distinction between named and anonymous styles. Consider a Microsoft Word document that consists of several paragraphs based on the default normal style. Suppose that one of the paragraphs is a quotation that needs to be indented and displayed in italics. If the document is part of a larger work that contains several quotations, it makes sense to define a special style for that purpose and apply it to all paragraphs that contain quotations. If the document is an early draft or is not likely to be updated, defining a style for one paragraph is overkill, and it would be more convenient to apply indentation and italics to the quotation itself.

In this analogy, specifying paragraph attributes directly is akin to setting the members of a property that returns an anonymous style. For example, to vertically center cell data within a particular grid column, modify the [VerticalAlignment](#) member of the column's [Style](#) property in the **C1DisplayColumnCollection Editor**.



Note that modifying an anonymous style is just like modifying a named style. Expand the desired [Style](#) object node in a property tree, and then select and edit one or more of its member properties.

Anonymous Style Inheritance

Just as one named style can inherit font, color, and formatting characteristics from another, an anonymous style in a [Split](#) object can inherit from its counterpart in the containing [C1TrueDBGrid](#) control. Similarly, an anonymous style in a [C1DisplayColumn](#) object can inherit from its counterpart in the containing [Split](#) object. Since the [C1TrueDBDropDown](#) control does not have a Splits collection, the anonymous styles of its [C1DisplayColumn](#) objects can inherit values from the control itself.

When a grid is first created, its [Style](#) property inherits all of its attributes from the built-in Normal style, which controls the appearance of all data cells. Any changes to the Normal style are propagated to all splits, and in turn to the columns within each split. However, change the appearance of all data cells within a [Split](#) or [C1DisplayColumn](#) object by modifying the members of its anonymous [Style](#) property.

Consider the following grid layout, which uses the default values of all built-in styles and contains two identical splits.

| Caption | | | |
|----------|----------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| Data | Data | Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

All of the subsequent examples use this layout as a starting point. For clarity, the examples use code to illustrate the relationships between style properties and the grid's display; however, you can perform the same operations at design time using the grid's collection editors.

Example 1 of 10: Inheriting from Containing Splits

Since the default values of all built-in styles are in effect, columns inherit from their containing splits, which in turn

inherit from the grid as a whole. Therefore, this statement affects not only data cells, but also all headers, footers, and caption bars. This statement has the same *visual* effect as changing the Normal style directly using the **C1TrueDBGrid Style Editor**; however, the built-in Normal style itself is not changed.

| Caption | | | |
|----------|----------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| ▶ Data | Data | ▶ Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

The following code inherits values from the containing splits:

To write code in Visual Basic

Visual Basic

```
Dim myfont As Font
myfont = New Font (Me.C1TrueDBGrid1.Styles("Normal").Font, FontStyle.Bold)
Me.C1TrueDBGrid1.Styles("Normal").Font = myfont
```

To write code in C#

C#

```
Font myfont;
myfont = new Font (this.c1TrueDBGrid1.Styles["Normal"].Font, FontStyle.Bold);
this.c1TrueDBGrid1.Styles["Normal"].Font = myfont;
```

Example 2 of 10: Affecting Only Data Cells in the First Split

In this example, only the data cells of the first split are affected. This is because the split caption, column headers, and column footers inherit their fonts from the built-in styles Caption, Heading, and Footing, respectively.

| Caption | | | |
|----------|----------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| ▶ Data | Data | ▶ Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

The following code affects data cells only in the first split:

To write code in Visual Basic

Visual Basic

```
Dim myfont As Font
myfont = New Font (Me.C1TrueDBGrid1.Splits(0).Style.Font, FontStyle.Bold)
Me.C1TrueDBGrid1.Splits(0).Style.Font = myfont
```

To write code in C#

C#

```
Font myfont;
myfont = new Font (this.c1TrueDBGrid1.Splits[0].Style.Font, FontStyle.Bold);
this.c1TrueDBGrid1.Splits[0].Style.Font = myfont;
```

Example 3 of 10: Affecting All Elements Only in the First Split

This example extends the previous one to render all elements of the first split in bold. In addition to the [Style](#) property, it is necessary to set the [CaptionStyle](#), [HeadingStyle](#), and [FooterStyle](#) properties.

| Caption | | | |
|-----------------|-----------------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| Data | Data | Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

The following code affects all elements only in the first split:

To write code in Visual Basic

Visual Basic

```
Dim myfont As Font
Dim myfont1 As Font
Dim myfont2 As Font
Dim myfont3 As Font

myfont = New Font (Me.C1TrueDBGrid1.Splits(0).Style.Font, FontStyle.Bold)
Me.C1TrueDBGrid1.Splits(0).Style.Font = myfont

myfont1 = New Font (Me.C1TrueDBGrid1.Splits(0).CaptionStyle.Font, FontStyle.Bold)
Me.C1TrueDBGrid1.Splits(0).CaptionStyle.Font = myfont1

myfont2 = New Font (Me.C1TrueDBGrid1.Splits(0).HeadingStyle.Font, FontStyle.Bold)
Me.C1TrueDBGrid1.Splits(0).HeadingStyle.Font = myfont2

myfont3 = New Font (Me.C1TrueDBGrid1.Splits(0).FooterStyle.Font, FontStyle.Bold)
Me.C1TrueDBGrid1.Splits(0).FooterStyle.Font = myfont3
```

To write code in C#

C#

```
Font myfont;
Font myfont1;
Font myfont2;
Font myfont3;

myfont = new Font (this.c1TrueDBGrid1.Splits[0].Style.Font, FontStyle.Bold);
this.c1TrueDBGrid1.Splits[0].Style.Font = myfont;
```

```
myfont1 = new Font (this.c1TrueDBGrid1.Splits[0].CaptionStyle.Font, FontStyle.Bold);
this.c1TrueDBGrid1.Splits[0].CaptionStyle.Font = myfont1;

myfont2 = new Font (this.c1TrueDBGrid1.Splits[0].HeadingStyle.Font, FontStyle.Bold);
this.c1TrueDBGrid1.Splits[0].HeadingStyle.Font = myfont2;

myfont3 = new Font (this.c1TrueDBGrid1.Splits[0].FooterStyle.Font, FontStyle.Bold);
this.c1TrueDBGrid1.Splits[0].FooterStyle.Font = myfont3;
```

Example 4 of 10: Affecting Only Data Cells in the First Column of the First Split

In this example, only the data cells of the first column of the first split are affected. This is because the column headers and column footers inherit their fonts from the built-in styles `Heading` and `Footing`, respectively.

| Caption | | | |
|-------------|----------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| Data | Data | Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

The following code affects data cells only in the first column of the first split:

To write code in Visual Basic

Visual Basic

```
Dim myfont As Font
myfont = New Font (Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).Style.Font,
FontStyle.Bold)
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).Style.Font = myfont
```

To write code in C#

C#

```
Font myfont;
myfont = new Font (this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].Style.Font,
FontStyle.Bold);
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].Style.Font = myfont;
```

Example 5 of 10: Affecting All Elements Only in the First Column of the First Split

This example extends the previous one to render all elements of the first column of the first split in bold. In addition to the [Style](#) property, it is necessary to set the [HeadingStyle](#) and [FooterStyle](#) properties.

| Caption | | | |
|----------|----------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| ▶ Data | Data | ▶ Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

The following code affects all elements only in the first column of the first split:

To write code in Visual Basic

Visual Basic

```
Dim myfont As Font
Dim myfont1 As Font
Dim myfont2 As Font

myfont = New Font (Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).Style.Font,
FontStyle.Bold)
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).Style.Font = myfont

myfont1 = New Font (Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).HeadingStyle.Font,
FontStyle.Bold)
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).HeadingStyle.Font = myfont1

myfont2 = New Font (Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).FooterStyle.Font,
FontStyle.Bold)
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).FooterStyle.Font = myfont2
```

To write code in C#

C#

```
Font myfont;
Font myfont1;
Font myfont2;

myfont = new Font (this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].Style.Font,
FontStyle.Bold);
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].Style.Font = myfont;

myfont1 = new Font (this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].HeadingStyle.Font,
FontStyle.Bold);
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].HeadingStyle.Font = myfont1;

myfont2 = new Font (this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].FooterStyle.Font,
FontStyle.Bold);
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].FooterStyle.Font = myfont2;
```

Example 6 of 10: Changing the BackColor of the Style Property

In the first example, setting the **Font** member of the grid's **Style** property affected the entire grid, including each caption bar, column header, and column footer. However, the same is not true of the **BackColor** and **ForeColor** properties. Since the built-in Caption, Heading, and Footing styles override both of these properties, only the data cells of the grid are displayed with a lavender background.

| Caption | | | |
|----------|----------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| Data | Data | Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

The following code changes the font member of the **Style** property:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Style.BackColor = System.Drawing.Color.Lavender
```

To write code in C#

C#

```
this.C1TrueDBGrid1.Style.BackColor = System.Drawing.Color.Lavender;
```

Example 7 of 10: Changing Only the Data Cells in the First Split

In this example, only the data cells of the first split are affected. This is because the split caption, column headers, and column footers inherit their background colors from the built-in styles Caption, Heading, and Footing, respectively.

| Caption | | | |
|----------|----------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| Data | Data | Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

The following code changes the data cells in only the first split:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).Style.BackColor = System.Drawing.Color.Lavender
```

To write code in C#

C#

```
this.C1TrueDBGrid1.Splits[0].Style.BackColor = System.Drawing.Color.Lavender;
```

Example 8 of 10: Changing Only the Data Cells in the First Column of the First Split

In this example, only the data cells of the first column of the first split are affected. This is because the column headers and column footers inherit their background colors from the built-in styles Heading and Footing, respectively.

| Caption | | | |
|----------|----------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| Data | Data | Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

The following code changes the data cells in only the first column of the first split:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).Style.BackColor =
System.Drawing.Color.Lavender
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].DisplayColumnsp[0].Style.BackColor =
System.Drawing.Color.Lavender;
```

Example 9 of 10: Setting the Alignment of C1DisplayColumn Objects

Setting the [HorizontalAlignment](#) property of a [C1DisplayColumn](#) object affects not only its data cells, but also its header and footer. The reason for this is that the default setting of the [HorizontalAlignment](#) property for the built-in Heading and Footing styles, which is inherited from the Normal style, is set to [AlignHorzEnum.General](#). For data cells, the general setting means that the underlying data type determines whether the cell text is left, center, or right aligned; for column headers and footers, the general setting means that the column's data cell alignment should be followed.

| Caption | | | |
|----------|----------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| Data | Data | Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

The following code sets the alignment of C1DisplayColumn objects:

To write code in Visual Basic

Visual Basic


```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).Style.HorizontalAlignment =
C1.Win.C1TrueDBGrid.AlignHorzEnum.Center
```

To write code in C#

```
C#
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].Style.HorizontalAlignment =
C1.Win.C1TrueDBGrid.AlignHorzEnum.Center;
```

Example 10 of 10: Setting the Alignment for Column Headers

This example illustrates the distinction between general and specific alignment for column headers and footers. If the [HorizontalAlignment](#) member of the [HeadingStyle](#) (or [FooterStyle](#)) property is not set to [AlignHorzEnum.General](#), then the header (or footer) is aligned independently of the data cells.

| Caption | | | |
|----------|----------|----------|----------|
| Split 0 | | Split 1 | |
| Column 0 | Column 1 | Column 0 | Column 1 |
| Data | Data | Data | Data |
| Data | Data | Data | Data |
| Footer 0 | Footer 1 | Footer 0 | Footer 1 |

The following code sets the alignment for column headers:

To write code in Visual Basic

```
Visual Basic
With Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0)
    .HeadingStyle.HorizontalAlignment = C1.Win.C1TrueDBGrid.AlignHorzEnum.Near
    .FooterStyle.HorizontalAlignment = C1.Win.C1TrueDBGrid.AlignHorzEnum.Far
    .Style.HorizontalAlignment = C1.Win.C1TrueDBGrid.AlignHorzEnum.Center
End With
```

To write code in C#

```
C#
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].HeadingStyle.HorizontalAlignment =
C1.Win.C1TrueDBGrid.AlignHorzEnum.Near;
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].FooterStyle.HorizontalAlignment =
C1.Win.C1TrueDBGrid.AlignHorzEnum.Far;
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].Style.HorizontalAlignment =
C1.Win.C1TrueDBGrid.AlignHorzEnum.Center;
```

Applying Styles to Cells

True DBGrid for WinForms provides three ways to control the display characteristics of individual cells:

| Control | Description |
|---------------------------|--|
| <i>By Status</i> | Each grid cell has a cell status that identifies its disposition (any combination of current, modified, part of a selected row, or part of a highlighted row). Using the AddCellStyle method, set style attributes that apply to any possible combination of cell status values. |
| <i>By Contents</i> | Specify a pattern (called a regular expression) that is used to perform pattern matching on cell contents. When the contents match the pattern supplied in the AddRegexCellStyle method, True DBGrid for WinForms will automatically apply pre-selected style attributes to the cell. |
| <i>By Custom Criteria</i> | Using the FetchCellStyle (or FetchRowStyle) event, make decisions about cell colors and fonts each time a cell (or row) is displayed. |

Use [Style](#) objects defined at design time as arguments to the [AddCellStyle](#) and [AddRegexCellStyle](#) methods. Or, create a temporary style in code and use it to specify one or more attributes.

The [FetchCellStyle](#) and [FetchRowStyle](#) events pass a temporary [Style](#) object as the final parameter. By setting its properties, control the appearance of the cell specified by the other event parameters.

In **True DBGrid for WinForms**, per-cell font and color control can only be achieved by writing code. However, by creating styles at design time, this code is kept to a minimum. To learn how to create named styles at design time, see [Using the C1TrueDBGrid Style Editor](#).

Specifying Cell Status Values

C1TrueDBGrid recognizes 16 distinct cell status values that are used in code to indicate the disposition of a cell. A cell status value is a combination of four separate conditions. These conditions are enumerations which have the *flag* attribute, which means that they can be combined with the *Or* operator:

| Condition | Description |
|---------------------|---|
| Current Cell | The cell is the current cell as specified by the Bookmark , Col , and SplitIndex properties. At any given time, only one cell can have this status. When the floating editor MarqueeStyle property setting is in effect, this condition is ignored. |
| Marquee Row | The cell is part of a highlighted row marquee. When the MarqueeStyle property indicates that the entire current row is to be highlighted, all visible cells in the current row have this additional condition set. |
| Updated Cell | The cell contents have been modified by the user but not yet written to the database. This condition is also set when cell contents have been modified in code with the Text or Value properties. |
| Selected Row | The cell is part of a row selected by the user or in code. The SelectedRowCollection contains a bookmark for each selected row. |

True DBGrid for WinForms defines the following constants corresponding to these cell conditions:

| Constant | Description |
|---|------------------------------|
| CellStyleFlag.CurrentCell | Applies to the current cell. |

| | |
|---|--|
| CellStyleFlag.MarqueeRow | Applies to cells in a highlighted row marquee. |
| CellStyleFlag.UpdatedCell | Applies to cells that have been modified. |
| CellStyleFlag.SelectedRow | Applies to cells in a selected row. |

True DBGrid for WinForms also defines the following constants, which are **not** meant to be combined with those listed earlier:

| Constant | Description |
|--|---|
| CellStyleFlag.AllCells | Applies to all cells. |
| CellStyleFlag.NormalCell | Applies to cells without status conditions. |

Use [CellStyleFlag.AllCells](#) to refer to all cells regardless of status. Use [CellStyleFlag.NormalCell](#) to refer to only those cells without any of the four basic cell conditions described earlier.

Applying Cell Styles by Status

Each cell in the **True DBGrid for WinForms** display has a status value which identifies its disposition (any combination of current, modified, part of a selected row, or part of a highlighted row). Using the [AddCellStyle](#) method, set style attributes that apply to any possible combination of cell status values. The [AddCellStyle](#) method is supported by the [C1TrueDBGrid](#), [C1TrueDBDropDown](#), [Split](#), and [C1DisplayColumn](#) objects, enabling the range of cells for which certain conditions apply to be controlled.

For each unique status combination, you can set the color, font, and picture attributes to be used for cells of that status. When a cell's status changes, **True DBGrid for WinForms** checks to see if any style property overrides are defined for that cell, and applies those attributes to the cell when it is displayed. [Style](#) objects are used to specify the color and font for a cell, as in the following example:

To write code in Visual Basic

Visual Basic

```
Dim S As New C1.Win.C1TrueDBGrid.Style()  
Dim myfont As Font  
  
myfont = New Font(S.Font, FontStyle.Bold)  
S.Font = myfont  
S.ForeColor = System.Drawing.Color.Red  
Me.C1TrueDBGrid1.AddCellStyle (C1.Win.C1TrueDBGrid.CellStyleFlag.CurrentCell, S)
```

To write code in C#

C#

```
C1TrueDBGrid.Style S = new C1.Win.C1TrueDBGrid.Style();  
Font myfont;  
  
myfont = new Font(S.Font, FontStyle.Bold);  
S.Font = myfont;  
S.ForeColor = System.Drawing.Color.Red;  
this.c1TrueDBGrid1.AddCellStyle(C1.Win.C1TrueDBGrid.CellStyleFlag.CurrentCell, S);
```

Here, a new temporary style object is created to specify the color and font overrides (red text, bold) to be applied to

the current cell throughout the entire grid. Since the style object's [BackColor](#) property is not set explicitly, the background color of the current cell is not changed.



| First | Last | Country |
|------------------|-----------|---------------|
| Isaac | Albeniz | Spain |
| Johann Sebastian | Bach | Germany |
| Samuel | Barber | United States |
| Bela | Bartok | Hungary |
| Ludwig van | Beethoven | Germany |
| Alban | Berg | Austria |

Also use styles defined at design time as arguments to the [AddCellStyle](#) method:

To write code in Visual Basic

Visual Basic

```
Dim S As Cl.Win.ClTrueDBGrid.Style
S = Me.ClTrueDBGrid1.Styles("RedBold")
Me.ClTrueDBGrid1.AddCellStyle(Cl.Win.ClTrueDBGrid.CellStyleFlag.CurrentCell, S)
```

To write code in C#

C#

```
Cl.Win.ClTrueDBGrid.Style S;
S = this.ClTrueDBGrid1.Styles("RedBold")
this.ClTrueDBGrid1.AddCellStyle(Cl.Win.ClTrueDBGrid.CellStyleFlag.CurrentCell, S);
```

The preceding example can be simplified since the [AddCellStyle](#) method accepts a style name as well as an actual style object:

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.AddCellStyle(Cl.Win.ClTrueDBGrid.CellStyleFlag.CurrentCell,
"RedBold")
```

To write code in C#

C#

```
this.ClTrueDBGrid1.AddCellStyle(Cl.Win.ClTrueDBGrid.CellStyleFlag.CurrentCell,
"RedBold");
```

All of the preceding examples cause the text of the current cell to appear in red and bold. However, it is important to note that the status [CellStyleFlag.CurrentCell](#) applies only to cells that have **only** this status. Thus, cells that are current but also updated ([CellStyleFlag.CurrentCell](#)+[CellStyleFlag.UpdatedCell](#)) will not be displayed in red and bold unless the following statement is executed:

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.AddCellStyle(Cl.Win.ClTrueDBGrid.CellStyleFlag.CurrentCell +
```

```
C1.Win.C1TrueDBGrid.CellStyleFlag.UpdatedCell, Me.C1TrueDBGrid1.Styles("RedBold"))
```

To write code in C#

C#

```
this.c1TrueDBGrid1.AddCellStyle(C1.Win.C1TrueDBGrid.CellStyleFlag.CurrentCell |  
C1.Win.C1TrueDBGrid.CellStyleFlag.UpdatedCell, this.c1TrueDBGrid1.Styles["RedBold"]);
```



Note: The current cell status is only honored when the [MarqueeStyle](#) property is **not** set to [MarqueeEnum.FloatingEditor](#). The floating editor marquee always uses the system highlight colors as determined by Control Panel settings.

Although this method of specifying cell conditions offers more control and flexibility, it also requires that additional code be written for some common cases.

Calls to [AddCellStyle](#) take effect immediately, and can be used for interactive effects as well as overall grid characteristics.

Applying Cell Styles by Contents

True DBGrid for WinForms can automatically apply colors and fonts to particular cells, based upon their displayed contents. To do so, provide a pattern, called a *regular expression* that the grid tests against the displayed value of each cell. Using the [AddRegexCellStyle](#) method, associate a regular expression with a set of style attributes, and then apply them to any possible combination of cell status values. The [AddRegexCellStyle](#) method is supported by the [C1TrueDBGrid](#), [C1TrueDBDropDown](#), [Split](#), and [C1DisplayColumn](#) objects, allowing the range of cells for which certain conditions apply to be controlled.

The [AddRegexCellStyle](#) method is similar to the [AddCellStyle](#) method, but it requires an additional argument for the regular expression string. As with [AddCellStyle](#), use either temporary or named styles. The following example uses a temporary style to display all cells in the first column that contain the string "Windows" in bold:

To write code in Visual Basic

Visual Basic

```
Dim S As New C1.Win.C1TrueDBGrid.Style()  
Dim myfont As Font  
  
myfont = New Font(S.Font, FontStyle.Bold)  
S.Font = myfont  
Me.C1TrueDBGrid1.AddRegexCellStyle (C1.Win.C1TrueDBGrid.CellStyleFlag.AllCells, S,  
"Computer")
```

To write code in C#

C#

```
C1TrueDBGrid.Style S = new C1.Win.C1TrueDBGrid.Style();  
Font myfont;  
  
myfont = new Font(S.Font, FontStyle.Bold);  
S.Font = myfont;  
this.c1TrueDBGrid1.AddRegexCellStyle (C1.Win.C1TrueDBGrid.CellStyleFlag.AllCells, S,  
"Computer");
```

This feature allows the implementation of "visual queries" that attach distinctive font or color attributes to cells that match a certain pattern.

| Company | LastName | FirstName |
|--------------------------------|------------|-----------|
| Microcomputer Consulting | Clark | Allen |
| Computer Engineering | Gordon | Alan |
| Software Designs | Quinn | Ann |
| Computer Communications | Wheeler | Alice |
| Microcomputers Unlimited | Ardmore | Beverly |
| Computer Applications | Fredericks | Carey |
| Micro Mechanics | Ziegler | Carl |
| Software Specialists | Elkins | David |

Applying Cell Styles by Custom Criteria

For cases where regular expressions are insufficient to express formatting requirements, use the [FetchCellStyle](#) event to customize fonts and colors on a per-cell basis. This event will only be fired for columns that have the [FetchStyle](#) property set to **True**.

For example, provide color coding for values that fall within a certain range. The following code assumes that the [FetchStyle](#) property is **True** for a single column of numeric data, and handles the [FetchCellStyle](#) event to display values greater than 1000 in blue:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_FetchCellStyle(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.FetchCellStyleEventArgs) Handles C1TrueDBGrid1.FetchCellStyle
    Dim N As Integer
    N = Val(Me.C1TrueDBGrid1(e.Row, e.Col))
    If N > 1000 Then
        e.CellStyle.ForeColor = System.Drawing.Color.Blue
    End If
End Sub
```

To write code in C#

C#

```
private void c1TrueDBGrid1_FetchCellStyle( object sender,
C1.Win.C1TrueDBGrid.FetchCellStyleEventArgs e)
{
    int N;
    N = (int) this.c1TrueDBGrid1[e.Row, e.Col];
    if ( N > 1000 )
    {
        e.CellStyle.ForeColor = System.Drawing.Color.Blue;
    }
}
```

The [Split](#), [Row](#), and [Col](#) properties identify which cell the grid is displaying. The [CellStyle](#) property conveys formatting

information from the application to the grid. Since the [CellStyle](#) property is a [Style](#) object, a cell's font characteristics can also be changed in the [FetchCellStyle](#) event:

To write code in Visual Basic

Visual Basic

```
If N > 1000 Then
    e.CellStyle.Font.Italic = True
Dim myfont As Font
myfont = New Font (e.CellStyle.Font, FontStyle.Italic)
If N > 1000 Then
    e.CellStyle.Font = myfont
```

To write code in C#

C#

```
if ( N > 1000 )
{
    e.CellStyle.Font.Italic = true
}
Font myfont;
myfont = new Font (e.CellStyle.Font, FontStyle.Italic);
if ( N > 1000 )
{
    e.CellStyle.Font = myfont;
}
```

The [FetchCellStyle](#) event can also be used to apply formatting to one cell based upon the values of other cells, or even other controls. For example, suppose that you want to:

- Make the cell text red in column 4 if column 1 minus column 2 is negative.
- Make the cell text bold in column 7 if it matches the contents of a text box.

In this case, set the [FetchStyle](#) property to **True** for columns 4 and 7, and handle the [FetchCellStyle](#) event as follows:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_FetchCellStyle(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.FetchCellStyleEventArgs) Handles C1TrueDBGrid1.FetchCellStyle
    Select Case e.Col
        Case 4
            Dim Col1 As Long, Col2 As Long
            Col1 = CLng(Me.C1TrueDBGrid1(e.Row, 1))
            Col2 = CLng(Me.C1TrueDBGrid1(e.Row, 2))
            If Col1 - Col2 < 0 Then
                CellStyle.ForeColor = System.Drawing.Color.Red
            Case 7
                Dim S As String
                S = Me.C1TrueDBGrid1(e.Row, 7).ToString()
                If S = TextBox1.Text Then
                    Dim myfont = New Font(CellStyle.Font, FontStyle.Bold)
```

```

        CellStyle.Font = myfont
    End If
    Case Else
        Debug.WriteLine ("FetchCellStyle not handled: " & e.Col)
    End Select
End Sub

```

To write code in C#

C#

```

private void c1TrueDBGrid1_FetchCellStyle( object sender,
Cl.Win.C1TrueDBGrid.FetchCellStyleEventArgs e)
{
    switch (e.Col)
    {
        case 4:
            long Col1, long Col2;
            Col1 = (long)this.c1TrueDBGrid1[e.Row, 1];
            Col2 = (long)this.c1TrueDBGrid1[e.Row, 2];
            if ( Col1 - Col2 < 0 )
                CellStyle.ForeColor = System.Drawing.Color.Red
            break;
        case 7:
            string S;
            S = this.c1TrueDBGrid1[e.Row, 7].ToString();
            if ( S == TextBox1.Text )
            {
                Font myfont = new Font(CellStyle.Font, FontStyle.Bold);
                CellStyle.Font = myfont;
            }
            break;
        default:
            Console.WriteLine ("FetchCellStyle not handled: " + e.Col);
    }
}

```

For efficiency reasons, only set [FetchStyle](#) to **True** for columns that you plan to handle in the [FetchCellStyle](#) event.



Note: The preceding examples use the [CellText](#) method for simplicity. However, the [CellText](#) and [CellValue](#) methods always create and destroy an internal clone of the dataset each time they are called, which may make them too inefficient to use in the [FetchCellStyle](#) event. To improve the performance of the grid's display cycle, try an unbound application. Unbound applications can access the underlying data source directly, which is generally faster than calling [CellText](#) or [CellValue](#).

To customize fonts and colors on a per-row instead of a per-cell basis, use the [FetchRowStyle](#) event, which will only be fired once per row for grids that have the [FetchRowStyles](#) property set to **True**. The syntax for this event is as follows:

To write code in Visual Basic

Visual Basic

```

Private Sub TDBGrid1_FetchRowStyle(ByVal sender As Object, ByVal e As
Cl.Win.C1TrueDBGrid.FetchRowStyleEventArgs) Handles C1TrueDBGrid1.FetchRowStyle

```


To write code in C#

C#

```
private void TDBGrid1_FetchRowStyle( object sender,
C1.Win.C1TrueDBGrid.FetchRowStyleEventArgs e)
```

Although the [FetchRowStyle](#) event can be used to implement an alternating row color scheme, an easier and more efficient way to accomplish the same task would be to use the [AlternatingRows](#) property, together with the built-in [EvenRow](#) and [OddRow](#) styles.

Cell Style Evaluation Order

The following list defines the order in which cell styles are applied relative to the anonymous styles of a grid, split, or column:

1. [Style](#) property, [C1TrueDBGrid](#) control. The default named parent of this anonymous style is [Normal](#).
2. [Style](#) property, [Split](#) object. By default, this anonymous style inherits from its [C1TrueDBGrid](#) control counterpart.
3. [EvenRowStyle](#) and [OddRowStyle](#) properties, [Split](#) object. By default, these anonymous styles inherit from their [C1TrueDBGrid](#) control counterparts, which in turn have default named parents of [EvenRow](#) and [OddRow](#). These properties apply only if the [AlternatingRows](#) property is **True**.
4. [Style](#) property, [C1DisplayColumn](#) object. By default, this anonymous style inherits from its [Split](#) object counterpart.
5. [FetchRowStyle](#) event. This event fires only if the [FetchRowStyles](#) property is **True** for a grid or split.
6. [SelectedStyle](#) property, [Split](#) object. By default, this anonymous style inherits from its [C1TrueDBGrid](#) control counterpart, which in turn has a default named parent of [Selected](#). This property applies only to selected rows; that is, rows whose bookmarks have been added to the [SelectedRowCollection](#) through code or user interaction.
7. [HighlightRowStyle](#) property, [Split](#) object. By default, this anonymous style inherits from its [C1TrueDBGrid](#) control counterpart, which in turn has a default named parent of [HighlightRow](#). This property applies only to highlighted rows, the current row in a grid or split whose [MarqueeStyle](#) property is set to [MarqueeEnum.HighlightRow](#) or [MarqueeEnum.HighlightRowRaiseCell](#).
8. [AddCellStyle](#) and [AddRegexCellStyle](#) methods, if called. Cell styles specified at the [C1DisplayColumn](#) object level have the highest priority, followed by those specified at the [Split](#) object and [C1TrueDBGrid](#) control levels. Within an object level, cell styles are tested in the order in which they were added in code. Cell styles do not inherit from one another; as soon as a match is found, testing stops.
9. [FetchCellStyle](#) event. This event fires only if the [FetchStyle](#) property is **True** for a [C1DisplayColumn](#) object.

Thus, you always have final control over the rendering of a cell via the [FetchCellStyle](#) event.

Applying Pictures to Grid Elements

In earlier versions of **True DBGrid for WinForms**, styles were used to specify font, color, and alignment attributes. This version extends the concept of styles to include background and foreground pictures, enabling adornments to be added to headers, footers, and caption bars, specify a background pattern for data cells, and render picture data in cells without having to populate a [ValueItems](#) object. The following properties of the [Style](#) object determine how pictures are displayed:

| Property | Description |
|---------------------------------|--|
| BackgroundImage | Sets/returns a style's background picture. |

| | |
|---|--|
| BackgroundPictureDrawMode | Controls how a style's background picture is displayed. |
| ForegroundImage | Sets/returns a style's foreground picture. |
| ForeGroundPicturePosition | Controls how a style's foreground picture is positioned. |

Since picture properties follow the same inheritance rules as other style attributes, any technique described earlier in this chapter also works with pictures. This means that pictures can be attached to a grid element using any of the following methods:

- Setting the [BackgroundImage](#) or [ForegroundImage](#) property of a built-in named style in the designer or in code.
- Setting the [BackgroundImage](#) or [ForegroundImage](#) property of an anonymous style in the designer or in code.
- Calling the [AddCellStyle](#) or [AddRegexCellStyle](#) method.
- Writing a handler for the [FetchCellStyle](#) or [FetchRowStyle](#) event.

Displaying Background Pictures

Use background pictures to customize static grid elements such as caption bars, column headers, and column footers. For example, the following code applies a colored gradient bitmap to the [BackgroundImage](#) member of the [Style](#) object returned by the grid's [CaptionStyle](#) property:

To write code in Visual Basic

Visual Basic

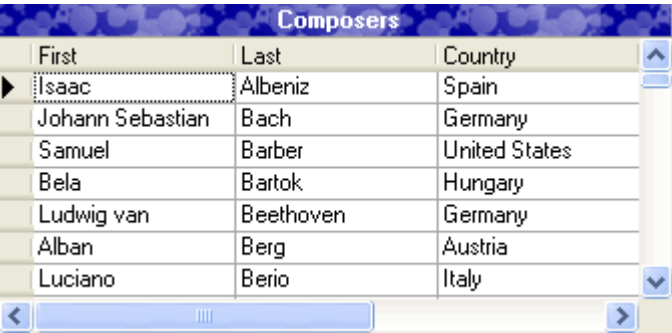
```
With Me.ClTrueDBGrid1.CaptionStyle
    .BackgroundImage = System.Drawing.Image.FromFile("c:\bubbles.bmp")
    .BackgroundPictureDrawMode =
C1.Win.ClTrueDBGrid.BackgroundPictureDrawModeEnum.Tile
    .ForeColor = System.Drawing.Color.White
    .Font = New Font(.Font, FontStyle.Bold)
End With
```

To write code in C#

C#

```
this.ClTrueDBGrid1.CaptionStyle.BackgroundImage =
System.Drawing.Image.FromFile(@"c:\bubbles.bmp");
this.ClTrueDBGrid.BackgroundPictureDrawMode =
C1.Win.ClTrueDBGrid.BackgroundPictureDrawModeEnum.Tile;
this.ClTrueDBGrid1.CaptionStyle.ForeColor = System.Drawing.Color.White;
this.ClTrueDBGrid1.CaptionStyle.Font = new Font(this.ClTrueDBGrid1.CaptionStyle.Font,
FontStyle.Bold);
```

This code also adjusts the color of the caption text and makes it bold, producing the following display.



Achieve the same effect at design time by editing either the built-in Caption style in the **C1TrueDBGrid Style Editor**, or the members of the [CaptionStyle](#) property in the Properties window.

By default, background pictures are centered within the associated grid element. Depending upon the height of the background bitmap, adjust the value of the [BackgroundPictureDrawMode](#) property to ensure that the entire area is filled. This property determines whether the picture is centered, tiled, or stretched to fit the entire area, as shown in the following table.

| Center | Tile | Stretch |
|---|--|---|
|  |  |  |

Also use background pictures within data cells to produce interesting visual effects. For example, the following patterns were designed to be replicated in adjacent rows.



By eliminating the record selector column, the dividing lines between data rows, and the column header dividers, these patterns can be used to produce the following display.

| Title | ISBN |
|---|---------------|
| Advanced Html : How to Do Cool Things With Your Web Site | 1-5561594-8-X |
| Developing Applications With Microsoft Office : Strategies for Designing, Develc | 1-5561566-5-0 |
| Developing Business Solutions With Microsoft Visual Basic and Microsoft Office | 1-5561589-9-8 |
| Developing International Software for Windows 95 and Windows Nt (Microsoft F | 1-5561584-0-8 |
| Developing Microsoft Excel 95 Solutions : With Visual Basic for Applications/Bo | 1-5561589-3-9 |
| Field Guide to Microsoft Powerpoint for Windows 95 | 1-5561584-1-6 |
| Field Guide to MS Access for Windows 95 | 1-5561587-5-0 |
| Field Guide to the Internet | 1-5561582-2-X |
| Hardcore Visual Basic/Book and Disk | 1-5561566-7-7 |
| Hitchhiker's Guide to Visual Basic & Sql Server -- Fourth Edition (Book and Disk) | 1-5561590-6-4 |
| Inside Odbc/Book and Cd-Rom | 1-5561581-5-7 |
| Inside Ole/Book and Cd-Rom (Microsoft Programming | 1-5561584-3-2 |
| Inside Visual C++ (Microsoft Programming/Book and Cd-Rom | 1-5561589-1-2 |
| Learn Microsoft Visual Basic 4 Now : The Complete Learning Solution for Visual | 1-5561590-5-6 |
| Mastering Microsoft Access/Book and Cd-Rom (Microsoft Mastering Series | 1-5561591-2-9 |
| Mastering Microsoft Visual Basic/Book and Cd Rom (Microsoft Mastering Series | 1-5561591-3-7 |

The trick is to insert an empty unbound column on the left to display the binder rings, as the following code sample demonstrates:

To write code in Visual Basic

Visual Basic

```
' Give the grid a flat appearance and remove its record selectors, row dividers, and
scroll bars.
With Me.C1TrueDBGrid1
    .InactiveStyle.ForeColor = System.Drawing.Color.White
    .RecordSelectors = False
    .RowDivider.Style = LineStyleEnum.None
    .RowHeight = 16
    .HScrollBar.Style = ScrollBarStyleEnum.None
    .VScrolBar.Style = ScrollBarStyleEnum.None
    .MarqueeStyle = MarqueeEnum.NoMarquee
End With

' Set the background pattern to be used by data cells in the default split (so as not
to disturb the Normal style).
With Me.C1TrueDBGrid1.Splits(0).Style
    .BackgroundImage = System.Drawing.Image.FromFile("paper.bmp")
    .BackgroundPictureDrawMode = BackgroundPictureDrawModeEnum.Tile
End With

' Create an empty unbound column on the left to hold the binder rings. Remove its
dividing lines and set the BackgroundBitmap property of its Style object.
Dim col as New C1TrueDBGrid.C1DataColumn()
Me.C1TrueDBGrid.Columns.InsertAt(0, col) Dim C As C1TrueDBGrid.C1DisplayColumn
C = Me.C1TrueDBGrid1.Splits(0).DisplayColumns(col)
With C
    .Width = 48
    .Visible = True
    .Style.BackgroundImage = System.Drawing.Image.FromFile("rings.bmp")
    .HeaderDivider = False
    .ColumnDivider.Style = LineStyleEnum.None
```

```

End With

' Scroll the unbound column into view.
Me.ClTrueDBGrid1.Col = 0

' Resize the Title column and remove its header dividers.
Set C = Me.ClTrueDBGrid1.Splits(0).DisplayColumns("Title")
With C
    .Width = 380
    .HeaderDivider = False
End With

' Use a small corner of the binder ring bitmap as the background of the column
headers, and adjust the font and text color accordingly.
Dim myfont As Font
With Me.ClTrueDBGrid1.HeadingStyle
    .BackgroundImage = System.Drawing.Image.FromFile("corner.bmp")
    .BackgroundPictureDrawMode = BackgroundPictureDrawModeEnum.Tile
    myfont = New Font(.Font, 10, FontStyle.Bold)
    .Font = myfont
    .ForeColor = System.Drawing.Color.White
End With

```

To write code in C#

```

C#

// Give the grid a flat appearance and remove its record selectors, row dividers, and
// scroll bars. Assume that the ScaleMode of the containing form is in pixels.
this.clTrueDBGrid1.InactiveStyle.ForeColor = System.Drawing.Color.White;
this.clTrueDBGrid1.RecordSelectors = false;
this.clTrueDBGrid1.RowDivider.Style = LineStyleEnum.None;
this.clTrueDBGrid1.RowHeight = 16;
this.clTrueDBGrid1.HScrollBar.Style = ScrollBarStyleEnum.None;
this.clTrueDBGrid1.VScrolBar.Style = ScrollBarStyleEnum.None;
this.clTrueDBGrid1.MarqueeStyle = MarqueeEnum.NoMarquee;

// Set the background pattern to be used by data cells in the default split (so as
// not to disturb the Normal style).
this.clTrueDBGrid1.Splits[0].Style.BackgroundImage =
System.Drawing.Image.FromFile("paper.bmp");
this.clTrueDBGrid1.Splits[0].Style.BackgroundPictureDrawMode =
BackgroundPictureDrawModeEnum.Tile;

// Create an empty unbound column on the left to hold the binder rings. Remove its
// dividing lines and set the BackgroundBitmap property of its Style object.
ClTrueDBGrid.ClDataColumn col = new ClTrueDBGrid.ClDataColumn();
this.ClTrueDBGrid.Columns.InsertAt(0, col);
ClTrueDBGrid.ClDisplayColumn C = this.clTrueDBGrid1.Splits[0].DisplayColumns[col];
    C.Width = 48;
C.Visible = true;
C.Style.BackgroundImage = System.Drawing.Image.FromFile["rings.bmp"];

```

```
C.HeaderDivider = false;
    C.ColumnDivider.Style = LineStyleEnum.None;

// Scroll the unbound column into view.
this.clTrueDBGrid1.Col = 0;





// Resize the Title column and remove its header dividers.
C = this.clTrueDBGrid1.Splits[0].DisplayColumns["Title"];
C.Width = 380;
C.HeaderDivider = false;



// Use a small corner of the binder ring bitmap as the background of the column
headers, and adjust the font and text color accordingly.
Font myfont;
this.clTrueDBGrid1.HeadingStyle.BackgroundImage =
System.Drawing.Image.FromFile("corner.bmp");
this.clTrueDBGrid1.HeadingStyle.BackgroundPictureDrawMode =
BackgroundPictureDrawModeEnum.Tile;
myfont = new Font(.Font, 10, FontStyle.Bold);
this.clTrueDBGrid1.HeadingStyle.Font = myfont;
this.clTrueDBGrid1.HeadingStyle.ForeColor = System.Drawing.Color.White;
```

Displaying Foreground Pictures

Use foreground pictures to add visual cues to static grid elements such as caption bars, column headers, and column footers. Foreground Pictures are specified by the [ForegroundImage](#) property of the [Style](#). Foreground pictures can be displayed beside some text or in place of it, but cannot be displayed over text.

Foreground pictures have the [ForegroundPicturePosition](#) property, which specifies where a foreground picture is situated in comparison to the cell text. The values and their representations are displayed as follows:

| Position | Display |
|-------------|---|
| Near |  |
| Far |  |
| LeftOfText |  |
| RightOfText |  |
| TopOfText |  |

| | |
|--------------|--|
| BottomOfText | <div>Cell Text </div> |
| PictureOnly | <div></div> |
| TextOnly | <div>Cell Text</div> |

Cell Editing Techniques

This section explains how to customize the behavior of cell editing in **True DBGrid for WinForms**. For text entry fields, write code in the grid's editing events, specify an input mask template, or display a drop-down text editor for long strings. To provide a list of choices for the user, use the [ValueItemCollection](#) object, the [C1TrueDBDropDown](#) control, or even an arbitrary intrinsic or third-party control.

How Cell Editing Works

True DBGrid for WinForms provides many features for customizing and controlling in-cell editing. The grid's default editing behavior depends on the setting of the [MarqueeStyle](#) property. If the floating editor marquee style is used, the editing behavior differs from that of other marquee styles. The following sections summarize **True DBGrid for WinForms**' editing behavior and state any exceptions that arise when using the floating editor.

For more information on the [MarqueeStyle](#) property, see [Highlighting the Current Row or Cell](#).

Initiating Cell Editing

A cell is either in display or edit mode. The [EditActive](#) property sets and returns the desired mode. Place the current cell in edit mode by setting [EditActive](#) to **True**, or end editing by setting it to **False**. The user may enter edit mode by clicking once on the current cell or by pressing the F2 key. A blinking text cursor (caret) will appear in the cell—at the beginning of the text when the cell is clicked and at the end when the F2 key is used. The [BeforeColEdit](#) event will be triggered when the cell enters edit mode. The [EditActive](#) property is **True** when the cell is in edit mode.

Floating Editor Differences: A blinking caret already exists at the beginning of the cell highlight even when in display mode. To enter edit mode, the user can click on any character location within the cell text to specify the text insertion point. The [BeforeColEdit](#) event is not triggered and the [EditActive](#) property is **False** until the user has made changes to the cell text.

Color and Wordwrap

In edit mode, the cell color is determined by the [ForeColor](#) and [BackColor](#) properties of the **EditorStyle** style object. The text being edited will wordwrap, regardless of the setting of the column style's [WrapText](#) property. If the text is too big to fit into the cell, a built-in drop-down edit control will automatically appear. For more information, see [Working with Text](#).

Floating Editor Differences: In edit mode, the text highlight disappears, and the cell color is the same as the normal cell color. The text being edited is wrapped only if the column style's **WrapText** property is **True**. The built-in drop-down edit control is not available.

Determining Modification Status

While editing is in progress, inspect the [DataChanged](#) property of the grid to determine whether the user has made any changes to the current row.

Set the grid's [DataChanged](#) property to **False** to exit editing, discard all changes to the current row, and refresh the current row display from the data source.

The icon in the record selector column of the current row reflects the status of the grid's [DataChanged](#) property. If [DataChanged](#) is **False**, a triangle-shaped arrow will be shown in the record selector column. If [DataChanged](#) is **True**, a pencil icon will appear instead.

Determining Cell Contents

While editing is in progress, the column's [Text](#) and [Value](#) properties contain the text the user currently sees in the modified row. Whenever the user presses a key, the [Change](#) event fires to notify the application that the user has just modified the current cell. However, the [Change](#) event does not mean the user is finished with the process, only that a single change has been made and the grid is still in edit mode.

The [Change](#) event does not fire when the grid is not in edit mode, such as when the contents of a cell are changed through code or when the user clicks a cell to cycle through [ValueItem](#) objects.

Terminating Cell Editing

The user completes the editing process by performing any of the following:

- Pressing the ENTER key.
- Pressing the ESC key.
- Moving to another cell with the arrow keys, the TAB key, or the mouse.
- Setting focus to another control on the form.

Handling Editing Events

The following sections describe the default editing behavior of **True DBGrid for WinForms** can be altered by responding to its events.

Standard Keystroke Events

True DBGrid for WinForms supports the standard keystroke events contained in the .NET environment:

| Event | Description |
|-----------------|--|
| KeyDown | Fired when the user presses a key. |
| KeyPress | Fired when the user presses an ANSI key. |
| KeyUp | Fired when the user releases a key. |

The **KeyDown** and **KeyUp** events trap all keys, including function keys, ALT and SHIFT keys, and numeric keypad keys. The **KeyPress** event only traps letters and numbers, punctuation marks and symbols, and editing keys such as TAB, ENTER, and BACKSPACE.

Use these events to restrict and modify user input as you would be done for any other intrinsic .NET control. For example, the following **KeyDown** event handler prevents the user from entering non-alphanumeric characters:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles C1TrueDBGrid1.KeyPress

    ' Cancel user key input if it is not a letter or a digit.
    If Not e.KeyChar.IsLetterOrDigit(e.KeyChar) Then
        e.Handled = True
    End If
```

```
End Sub
```

To write code in C#

C#

```
private void CltrueDBGrid1_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
{
    // Cancel user key input if it is not a letter or a digit.
    if (! e.KeyChar.IsLetterOrDigit(e.KeyChar])
    {
        e.Handled = true ;
    }
}
```

For more information on these or any other native .NET events see MSDN or .NET help.

Column Editing Events

True DBGrid for WinForms provides full control over the cell editing process with the following events, listed in the order in which they occur during a successful editing attempt:

| Event | Description |
|-------------------------------|---|
| BeforeColEdit | Fired upon an attempt to edit column data. |
| ColEdit | Fired when the current cell enters edit mode. |
| AfterColEdit | Fired after column data is edited. |

Use the [BeforeColEdit](#) event to control the editability of cells on a per-cell basis, or to translate the initial keystroke into a default value.

The [ColEdit](#) event signals that the current cell has entered edit mode; the [AfterColEdit](#) event signals that edit mode was terminated. Use these two events to provide additional feedback while editing is in progress:

To write code in Visual Basic

Visual Basic

```
Private Sub ClTrueDBGrid1_ColEdit(ByVal sender As Object, ByVal e As
Cl.Win.ClTrueDBGrid.ColEventArgs) Handles ClTrueDBGrid1.ColEdit
    Select Case e.Columns.DataColumn.Caption
        Case "Code"
            Me.Label1.Text = "Enter 4-digit company code"
        Case "Description"
            Me.Label1.Text = "Enter full company name"
    End Select
End Sub

Private Sub ClTrueDBGrid1_AfterColEdit (ByVal sender As Object, ByVal e As
Cl.Win.ClTrueDBGrid.ColEventArgs) Handles ClTrueDBGrid1.AfterColEdit

    ' Clear editing instructions.
```

```
Me.Label1.Text = ""  
End Sub
```

To write code in C#

```
C#  
  
private void CltrueDBGrid1_ColEdit(object sender, Cl.Win.ClTrueDBGrid.ColEventArgs e)  
{  
    switch(e.Columns.DataColumn.Caption)  
    {  
        Case "Code":  
            this.Label1.Text = "Enter 4-digit company code";  
            break;  
        Case "Description":  
            this.Label1.Text = "Enter full company name";  
            break;  
    }  
}  
  
private void ClTrueDBGrid1_AfterColEdit(object sender,  
Cl.Win.ClTrueDBGrid.ColEventArgs e)  
{  
    // Clear editing instructions.  
    this.Label1.Text = "";  
}
```

Changing Cell Contents with a Single Keystroke

You can use the [BeforeColEdit](#) event to customize the editing behavior of **True DBGrid for WinForms** controls. [BeforeColEdit](#) is fired before any other editing events occur, which provides the opportunity to do virtually anything desired before editing begins. For example, cancel the edit request and override the built-in text editor with your own drop-down list box.

A **True DBGrid for WinForms** control enters edit mode in one of four ways:

- If the user clicks on the current cell with the mouse, editing begins with the current cell contents.
- If the user presses the F2 key, editing also begins using the current cell contents.
- If the user begins typing, the typed character replaces the contents of the cell and editing begins.
- You can set the [EditActive](#) property in your code to force editing to begin.

The [BeforeColEdit](#) event fires in the first three cases, but *not* in the last case, since **True DBGrid for Winforms** assumes you will never want to cancel a request made from code.

To differentiate a user's edit request based upon whether he or she used the mouse or the keyboard to start editing, set [BeforeColEdit](#) to [KeyChar](#), which will be zero if the user clicked on the cell with the mouse, and will be an ASCII character if the user typed a character to begin editing.

When [BeforeColEdit](#) is fired, the ASCII character has not yet been placed into the current cell, so if editing in [BeforeColEdit](#) is cancelled, the ASCII key is discarded. This leads to an interesting technique.

Assume a Boolean field called *Done* exists, and its [NumberFormat](#) property is set to specify Yes/No as the display format. Further assume that, when the user presses Y or N, the cell contents change immediately instead of entering edit mode. This process is accomplished in [BeforeColEdit](#) as follows:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_BeforeColEdit(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.BeforeColEditEventArgs) Handles C1TrueDBGrid1.BeforeColEdit
    With Me.C1TrueDBGrid1.Columns(e.ColIndex)

        ' If this isn't the "Done" column, or if the user clicked with the mouse,
        then simply continue.
        If .DataField <> "Done" Or e.KeyChar = Chr(0) Then Exit Sub

        ' Cancel normal editing and set the field to the proper result based upon
        KeyChar. Beep if an invalid character was typed.
        e.Cancel = True
        Select Case UCase(e.KeyChar)
            Case "Y"
                .Value = -1
            Case "N"
                .Value = 0
            Case Else
                Beep()
        End Select
    End With
End Sub
```

To write code in C#

C#

```
private void C1TrueDBGrid1_BeforeColEdit( object sender,
C1.Win.C1TrueDBGrid.BeforeColEditEventArgs e)
{
    C1.Win.C1DataColumn col = e.Column.DataColumn;

    // If this isn't the "Done" column, or if the user clicked with the mouse, then
    simply continue.
    if (col.DataField != "Done" || e.KeyChar == 0 ) return;

    // Cancel normal editing and set the field to the proper result based upon
    KeyChar. Beep if an invalid character was typed.
    e.Cancel = true;
    switch (e.KeyChar. .ToUpper())
    {
        case "Y";
            Col.Value = -1;
            break;
        case "N";
            Col.Value = 0;
        default;;
            Beep();
    }
}
```

Note that the event handler terminates when [KeyChar](#) is zero, so mouse editing is still permitted.

Working with Text

This section briefly describes the properties related to text editing.

Limiting the Size of Data Entry Fields

Use the [DataWidth](#) property of a [C1DataColumn](#) object to restrict the number of characters the user can enter. Setting this property to zero imposes no limits.

Providing a Drop-Down Edit Control for Long Fields

Whenever the user attempts to edit cell text that is too big to fit within the cell, the grid will automatically activate a multiple-line drop-down text editor. While editing, text in the drop-down edit control will be wrapped regardless of the setting of the column style's [WrapText](#) property. The drop-down text editor can be turned off and editing can be forced to occur within cell boundaries by setting the grid's [EditDropDown](#) property to **False** (the default is **True**). The drop-down text editor is **not** available if the grid's [MarqueeStyle](#) property is set to [MarqueeEnum.FloatingEditor](#). The following code uses the grid's built-in column button feature to activate the drop-down edit control to modify the cell data in the *Comments* column:

To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles MyBase.Load
        With Me.C1TrueDBGrid1
            .MarqueeStyle = MarqueeEnum.SolidCellBorder
            .Splits(0).DisplayColumns("Comments").Button = True

            ' Redundant since default = True.
            .EditDropDown = True
        End With
    End Sub

Private Sub C1TrueDBGrid1_ButtonClick(ByVal sender As Object, ByVal e As
    C1.Win.C1TrueDBGrid.ColEventArgs) Handles C1TrueDBGrid1.ButtonClick
    ' Place the cell into edit mode.
    Me.C1TrueDBGrid1.EditActive = True
End Sub
```

To write code in C#

C#

```
private void Form1_Load(System.Object sender, System.EventArgs e)
{
    C1TrueDBGrid1.MarqueeStyle = MarqueeEnum.SolidCellBorder;
    C1TrueDBGrid1.Splits[0].DisplayColumns["Comments"].Button = true;
}
```

```
// Redundant since default = true.
C1TrueDBGrid1.EditDropDown = true;
}

private void C1TrueDBGrid1_ButtonClick(object sender,
C1.Win.C1TrueDBGrid.ColEventArgs e)
{
    // Place the cell into edit mode.
    this.c1TrueDBGrid1.EditActive = true;
}
```

If the current cell is in the *Comments* column, initiate editing either by clicking on the current cell or by clicking the built-in button.

Selecting and Replacing Text

True DBGrid for WinForms supports the standard text selection properties found in many `TextBox` type controls:

| Property | Description |
|---------------------------------|---|
| SelectionLength | Sets/returns the length of the selected text. |
| SelectionStart | Sets/returns the start position of the selected text. |
| SelectedText | Sets/returns the selected text. |



Note: These properties are only effective when the grid is in edit mode, that is, when its [EditActive](#) property is **True**.

Input Masking

Use the [NumberFormat](#) property to control the display format of column data. If users need to edit a formatted column, it is desirable to maintain a consistent format during the editing process. **True DBGrid for WinForms** provides an [EditMask](#) property that optionally works in concert with the [NumberFormat](#) property to ensure consistent data entry.

Specifying an Input Mask for a Column

The [EditMask](#) property of the [C1DataColumn](#) object is used to specify an input mask template for end-user data entry. The input mask string is composed of special characters that represent either an input character that the user must enter or a literal character that will be skipped over on input. Valid template characters are as follows:

The [EditMask](#) must be a string composed of the following symbols:

1. Wildcards

- 0 digit
- 9 digit or space
- # digit or sign
- L letter
- ? letter or space

- A letter or digit
- a letter, digit, or space
- & any character

2. Localized characters

- . localized decimal separator
- , localized thousand separator
- : localized time separator
- / localized date separator

3. Command characters

- \ next character is taken as a literal
- > translate letters to uppercase
- < translate letters to lowercase

For example:

To write code in Visual Basic

Visual Basic

```
' Set the mask so the user can enter a phone number, with optional area code, and a
state in capitals.
Me.ClTrueDBGrid1.Columns(0).EditMask = "(###) 000-0000 St\ate\ : >LL"
```

To write code in C#

C#

```
// Set the mask so the user can enter a phone number, with optional area code, and a
state in capitals.
this.ClTrueDBGrid1.Columns[0].EditMask = "(###) 000-0000 St\\ate\\ : >LL";
```

Using an Input Mask for Formatting

Whereas the [EditMask](#) property is used to specify an input mask for *data entry*, the [NumberFormat](#) property is used to specify the *display format* of data in a grid cell. If the [NumberFormat](#) property of the column is not specified, the grid simply displays the cached text (stripped of literals) as is; if the [NumberFormat](#) property is specified, the grid sends the cached text to the display formatter.

Since it is common for the input and display formats to be the same, the [NumberFormat](#) property has an *Edit Mask* option. If this option is selected, then the [EditMask](#) property setting will be used for both data input and display. However, the input and display formats need not be the same, so a [NumberFormat](#) option that differs from the [EditMask](#) property can be selected.

Controlling How Masked Input is Updated

Normally, after the user finishes editing a cell in a column which has its [EditMask](#) property set, **True DBGrid for Winforms** caches the modified cell text, but any literal characters in the input mask template will be stripped from the modified cell text beforehand. However, this behavior can be overridden with the [EditMaskUpdate](#) property.

By default, the [EditMaskUpdate](#) property is **False**. This means that when the modified cell text is updated to the database, the grid sends the cached text (stripped of literals), not the formatted text displayed in the cell. Override this default behavior by setting the [EditMaskUpdate](#) property to **True**, which causes the cached text to be formatted according to the [EditMask](#) property before being updated to the database.

Therefore, it is important to set [EditMaskUpdate](#) properly to ensure that the correct data is sent to the database for update.

In-Cell Buttons

True DBGrid for WinForms supports a variety of in-cell button options for the current cell or for all cells within specified columns. Use in-cell buttons to indicate that a list of choices is available, to perform a command associated with the contents of the cell, or to display an arbitrary control or form for editing.

Enabling the In-Cell Button

To enable the in-cell button for a [C1DisplayColumn](#) object, set its [Button](#) property to **True** in code:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).Button = True
```

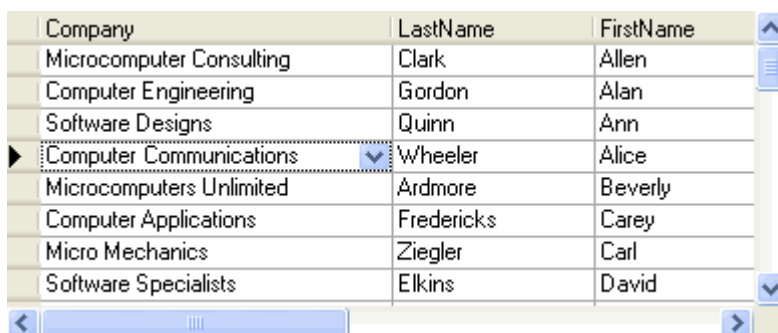
To write code in C#

C#

```
this.C1TrueDBGrid1.Splits[0].DisplayColumns[0].Button = true;
```

The [Button](#) property is also enabled when the column's [DropDown](#) property is set to the name of a [C1TrueDBDropDown](#) control, or when the [Presentation](#) property of the associated [ValueItemCollection](#) object is set to one of the combo box options.

By default, the in-cell button displays only for the current cell, as shown in the following image:



| Company | LastName | FirstName |
|--------------------------|------------|-----------|
| Microcomputer Consulting | Clark | Allen |
| Computer Engineering | Gordon | Alan |
| Software Designs | Quinn | Ann |
| Computer Communications | Wheeler | Alice |
| Microcomputers Unlimited | Ardmore | Beverly |
| Computer Applications | Fredericks | Carey |
| Micro Mechanics | Ziegler | Carl |
| Software Specialists | Elkins | David |

However, by setting the column's [ButtonAlways](#) property to **True**, you can force the in-cell button to be displayed in every row:

| Company | LastName | FirstName |
|--------------------------|------------|-----------|
| Microcomputer Consulting | Clark | Allen |
| Computer Engineering | Gordon | Alan |
| Software Designs | Quinn | Ann |
| Computer Communications | Wheeler | Alice |
| Microcomputers Unlimited | Ardmore | Beverly |
| Computer Applications | Fredericks | Carey |
| Micro Mechanics | Ziegler | Carl |
| Software Specialists | Elkins | David |

Rendering Cells as Command Buttons

To render the current cell as a non-editable command button within a `C1DisplayColumn` object, set its `ButtonText` property to **True** in code:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).ButtonText = True
```

To write code in C#

C#

```
this.C1TrueDBGrid1.Splits[0].DisplayColumns[0].ButtonText = true;
```

When a cell within the column receives focus, it is rendered as a standard Windows command button using the cell text as the caption. The cell text is not centered automatically, but respects the column's horizontal and vertical alignment settings:

| Company | LastName | FirstName |
|--------------------------|------------|-----------|
| Microcomputer Consulting | Clark | Allen |
| Computer Engineering | Gordon | Alan |
| Software Designs | Quinn | Ann |
| Computer Communications | Wheeler | Alice |
| Microcomputers Unlimited | Ardmore | Beverly |
| Computer Applications | Fredericks | Carey |
| Micro Mechanics | Ziegler | Carl |
| Software Specialists | Elkins | David |

If both the `Button` and `ButtonText` properties are **True**, the `ButtonText` property takes precedence.

As with the default in-cell button, set the column's `ButtonAlways` property to **True** to force all of its cells to be displayed as command buttons. Only the current cell is drawn with a focus rectangle, however:

| Company | LastName | FirstName |
|--------------------------|------------|-----------|
| Microcomputer Consulting | Clark | Allen |
| Computer Engineering | Gordon | Alan |
| Software Designs | Quinn | Ann |
| Computer Communications | Wheeler | Alice |
| Microcomputers Unlimited | Ardmore | Beverly |
| Computer Applications | Fredericks | Carey |
| Micro Mechanics | Ziegler | Carl |
| Software Specialists | Elkins | David |

Detecting In-Cell Button Clicks

The [ButtonClick](#) event is provided so that code can respond when the user clicks the in-cell button. Its syntax is as follows:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_ButtonClick(ByVal sender As Object, ByVal e As C1.Win.C1TrueDBGrid.ColEventArgs) Handles C1TrueDBGrid1.ButtonClick
```


To write code in C#

C#

```
private void C1TrueDBGrid1_ButtonClick( object sender, C1.Win.C1TrueDBGrid.ColEventArgs e)
```

In-cell buttons always fire this event when clicked, regardless of whether they were enabled by the [Button](#) or [ButtonText](#) properties. An example of the [ButtonClick](#) event was presented earlier in the section [Working with Text](#).

Customizing the In-Cell Button Bitmap

By default, **True DBGrid for WinForms** uses a down arrow for the in-cell button. 

However, the button bitmap can be changed for a [C1DisplayColumn](#) object at design time by setting the [ButtonPicture](#) property in code:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns(0).ButtonPicture = System.Drawing.Image.FromFile("dollar.bmp")
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Columns[0].ButtonPicture = System.Drawing.Image.FromFile("dollar.bmp");
```

The grid automatically draws the edges corresponding to the button's up/down states as appropriate, so only the

interior image of the button needs to be provided.

| Company | LastName | FirstName |
|--------------------------|------------|-----------|
| Microcomputer Consulting | Clark | Allen |
| Computer Engineering | Gordon | Alan |
| Software Designs | Quinn | Ann |
| Computer Communications | Wheeler | Alice |
| Microcomputers Unlimited | Ardmore | Beverly |
| Computer Applications | Fredericks | Carey |
| Micro Mechanics | Ziegler | Carl |
| Software Specialists | Elkins | David |

Drop-Down Controls

True DBGrid for WinForms offers a wide variety of built-in controls and programming constructs that enable you to implement virtually any kind of drop-down cell editing interface. Use the [ValueItems](#) object and its collection of [ValueItem](#) objects to provide a simple pick list, or the [C1TrueDBDropDown](#) control to implement a data-aware multicolumn combo box. Arbitrary Visual Basic or third-party controls can be used to perform specialized editing functions.

Using the Built-In Combo Box

The [C1DataColumn](#) object's [ValueItems](#) object optionally provides a built-in combo box interface that works in concert with its automatic data translation features. By default, the [Presentation](#) property is set to [PresentationEnum.Normal](#), and the usual cell editing behavior is in effect for textual data. However, if the [Presentation](#) property is set to either [PresentationEnum.ComboBox](#) or [PresentationEnum.SortedComboBox](#), then cells in the affected column display the in-cell button upon receiving focus. When the user clicks the in-cell button, a drop-down combo box appears.

| Company | Country |
|-----------------------------------|----------------|
| Maple Leaf Systems | Canada |
| Her Majesty's Software | Canada |
| Software Mart | United Kingdom |
| Far East Distributors | United States |
| Outback Software, Inc. | Japan |
| Northwest Purchasing Agents, Inc. | Australia |

The drop-down combo box contains one item for each member of the [ValueItemCollection](#) object. If the collection's [Translate](#) property is **True**, then the [DisplayValue](#) text is used for the combo box items; if it is **False**, then the [Value](#) text is used.

True DBGrid for WinForms automatically sizes the drop-down combo box to fit the width of the column in which it is displayed. The height of the combo box is determined by the number of items in the collection and the [MaxComboItems](#) property. If the number of items is less than or equal to [MaxComboItems](#), which has a default value of 5, and then all value items will be shown. If the number of items exceeds [MaxComboItems](#), only [MaxComboItems](#) will be shown, but a scroll bar will appear at the right edge of the combo box to allow users to bring the other items into view.

Detecting Built-In Combo Box Selections

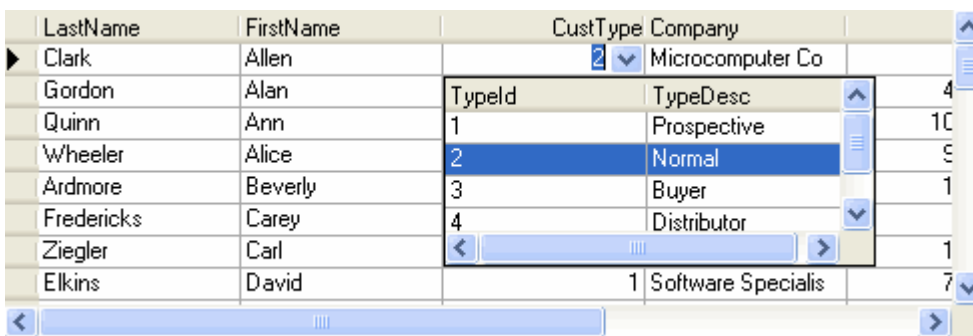
The [ComboSelect](#) event is fired when the user selects an item from the built-in combo box. This event is useful for determining the contents of the cell before the user exits edit mode.

Since the items displayed in the built-in combo box are often the only allowable values for the underlying data source, you may need to prevent your users from typing in the cell after making a selection. By setting the [C1DisplayColumn](#) property [DropDownList](#) equal to **True**, the attached [C1TrueDBDropDown](#) control will now be limited to use only as a list box. No new values or changes will be allowed in the drop-down and so the underlying database cannot be updated with false information.

Using the C1TrueDBDropDown Control

The built-in drop-down combo box described in the preceding example is most useful when the allowable values are both known in advance and relatively few in number. A large collection of [ValueItem](#) objects can be unwieldy to maintain in the designer, and requires substantial coding to set up. Moreover, the built-in combo box cannot be bound to a data control and be populated automatically.

Using the techniques outlined later in this chapter, set up a secondary [C1TrueDBGrid](#) control to be used as a drop-down. However, to display a list of values from another data source, the [C1TrueDBDropDown](#) control offers a more elegant solution, as it was designed explicitly for that purpose and can be set up entirely at design time.



To use the drop-down control, set the [DropDown](#) property of a grid column to the [C1TrueDBDropDown](#) control either in the designer or in code. At run time, when the user clicks the in-cell button for that column, the [C1TrueDBDropDown](#) control will appear below the grid's current cell. If the user selects an item from the drop-down control, the grid's current cell is updated.

Since the [C1TrueDBDropDown](#) control is a subset of [C1TrueDBGrid](#), it shares many of the same properties, methods, and events. However, the following two properties are specific to the [C1TrueDBDropDown](#) control:

| Property | Description |
|-------------------------------|--|
| ValueMember | This property specifies the drop-down column used to update the associated grid column when a selection is made. |
| DisplayMember | This property specifies the name of the drop-down column to be used for incremental search. |

When a [C1TrueDBDropDown](#) control becomes visible, its [DropDownOpen](#) event fires. Similarly, when the user makes a selection or the control loses focus, its [DropDownClose](#) event fires.

Automatic Data Translation with C1TrueDBDropDown

Suppose a grid drop-down box is needed using data that contains a value and a corresponding text representation, as in the following image:

| TypeId | TypeDesc |
|--------|-------------|
| 1 | Prospective |
| 2 | Normal |
| 3 | Buyer |
| 4 | Distributor |
| 5 | Other |

In this situation, you may not want the user to see the somewhat ambiguous *TypeId*, but instead want the more understandable *TypeDesc* to show in the drop-down. The [ValueTranslate](#) property automatically maps the *TypeId* value to the *TypeDesc* representation. In this way, when the user accesses the drop-down, it will display the *TypeDesc* text.

Using an Arbitrary Drop-Down Control

Normally, **True DBGrid for WinForms**' default editing behavior is sufficient for most applications. In some cases, however, you may want to customize this behavior. One valuable technique is to use a drop-down list or combo box, or even another **True DBGrid for WinForms** control, to allow selection from a list of possible values. This is easy to do with **True DBGrid for WinForms** using virtually any Visual Studio or third-party control. The general approach follows, and a working example is given in [Tutorial 9: Attaching an Arbitrary Drop-Down Control to a Grid Cell](#).

In general, displaying a drop-down list or combo instead of the standard **True DBGrid** editor involves the following steps:

1. **True DBGrid for WinForms** fires the [BeforeColEdit](#) event each time the user wants to edit a cell. To override the default editing process, cancel **C1TrueDBGrid**'s default editor by setting the *Cancel* parameter to **True**. Put code in [BeforeColEdit](#) to display the editing control you wish to show instead. Typically, you place the substitute editing control or drop-down on the same form as the grid, but make it invisible until you need it.
2. When [BeforeColEdit](#) is triggered, there are five properties and one method that can be used to determine the exact coordinates of the cell that is to be edited. The properties are **Left** (applies to grid and column), **Top** (grid and column), [CellTop](#) (column only, used with multiple line displays), [Width](#) (column only), and [RowHeight](#) (grid only). The method is [RowTop](#) (grid only). Use these properties and method to position the custom editing control or drop-down relative to a grid cell. For example, place a `ListBox` control at the right edge of a cell and align its top border with that of the cell using the following code:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_BeforeColEdit(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.BeforeColEditEventArgs) Handles C1TrueDBGrid1.BeforeColEdit
    Dim r As Rectangle =
Me.C1TrueDBGrid1.Splits(0).GetCellBounds(Me.C1TrueDBGrid1.Row, e.ColIndex)
    r = Me.C1TrueDBGrid1.RectangleToScreen(r)
    r = Me.RectangleToClient(r)
    Me.ListBox1.Left = r.Left
    Me.ListBox1.Top = r.Bottom
End Sub
```

To write code in C#

C#

```
private void c1TrueDBGrid1_BeforeColEdit(object sender,
C1.Win.C1TrueDBGrid.BeforeColEditEventArgs e)
```

```
{
    Rectangle r =
this.clTrueDBGrid1.Splits[0].GetCellBounds(this.clTrueDBGrid1.Row, e.ColIndex);
    r = this.clTrueDBGrid1.RectangleToScreen(r);
    r = this.RectangleToClient(r);
    this.ListBox1.Left = r.Left;
    this.ListBox1.Top = r.Bottom;
}
```

3. Put code in the drop-down or combo box which completes the editing process by assigning the selected value to the [Text](#) or [Value](#) property of the column being edited.

This method does not work, however, when the grid's [MarqueeStyle](#) property is set to the value of [MarqueeEnum.FloatingEditor](#). When the floating editor marquee is used, the [BeforeColEdit](#) event does not fire until the cell has been changed by the user. However, use the built-in column button feature to activate the drop-down box as described in the next section.

For illustrations of other [MarqueeStyle](#) settings, see [Highlighting the Current Row or Cell](#). An example of dropping down a Visual Basic ListBox control from a grid cell is given in [Tutorial 9: Attaching an Arbitrary Drop-Down Control to a Grid Cell](#).

Using the Built-In Column Button

An alternative way to drop-down a control from a cell is to use **True DBGrid for WinForms'** built-in column button feature. If a column's [Button](#) property is set to **True**, a button will be displayed at the right edge of the current cell when it is in that column. Clicking the button fires the grid's [ButtonClick](#) event. Drop-down a control from the cell using code inside the [ButtonClick](#) event. Also use this event to trigger any action or calculation inside the cell.

For more information, see [In-Cell Buttons](#).

True DBGrid for WinForms Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos which may make use of other development tools included with the ComponentOne Studio Enterprise.

Please refer to the pre-installed product samples through the following path:

Documents\ComponentOne Samples\WinForms

Click one of the following links to view a list of **True DBGrid for WinForms** samples:

Visual Basic Samples

| Sample | Description |
|--------------------|---|
| AggreGateFooter | Using notifications to customize the grids footer. This sample uses the C1TrueDBGrid control. |
| AutoFilter | Using C1TrueDBDropDown in the filter bar. This sample uses the C1TrueDBGrid and C1TrueDBDropDown controls. |
| CustomFiltering | Roll your own filtering for the grid. This sample uses the C1TrueDBGrid control. |
| CustomSorting | Roll your own sorting. This sample uses the C1TrueDBGrid control. |
| DateTimePicker | How to use a DateTimePicker control in the grid for date columns. This sample uses the C1TrueDBGrid control. |
| FindRow | How to find a row in the underlying datasource. This sample uses the C1TrueDBGrid control. |
| HyperLink | Add hyperlinks to cells. This sample uses the C1TrueDBGrid control. |
| IncrementalSearch | Add incremental search to the grid. This sample uses the C1TrueDBGrid control. |
| MultipleLayouts | How to store multiple layout files. This sample uses the C1TrueDBGrid control. |
| MultipleSelection | Select or deselect rows when you click a row. |
| SettingCellToNull | How to set the underlying datasource value to null. This sample uses the C1TrueDBGrid control. |
| ToggleGroupRows | Programmatically expand/collapse rows in a grouped grid. This sample uses the C1TrueDBGrid control. |
| TriStateCheckBox | How to add a tristate check box to the grid. This sample uses the C1TrueDBGrid control. |
| UsingC1TDBDropDown | How to use C1TrueDBDropDown to map IDs to Names. This sample uses the C1TrueDBGrid and C1TrueDBDropDown controls. |
| Zoom | Change the size of the grid. This sample uses the C1TrueDBDropDown control. |

C# Samples

| Sample | Description |
|-----------------|--|
| AggreGateFooter | Using notifications to customize the grids footer. This sample uses the C1TrueDBGrid control. |
| AutoFilter | Using C1TrueDBDropDown in the filter bar. This sample uses the C1TrueDBGrid and C1TrueDBDropDown controls. |

| | |
|----------------------|---|
| CustomFiltering | Roll your own filtering for the grid. This sample uses the C1TrueDBGrid control. |
| CustomSorting | Roll your own sorting. This sample uses the C1TrueDBGrid control. |
| DateTimePicker | How to use a datetimepicker control in the grid for date columns. This sample uses the C1TrueDBGrid control. |
| FilterDefinitionTdbg | Uses the FilterDefinition property to save/load custom filters in code. This sample application enables users to apply one of a few pre-defined filters. |
| FindRow | How to find a row in the underlying datasource. This sample uses the C1TrueDBGrid control. |
| HyperLink | Add hyperlinks to cells. This sample uses the C1TrueDBGrid control. |
| IncrementalSearch | Add incremental search to the grid. This sample uses the C1TrueDBGrid control. |
| MultipleLayouts | How to store multiple layout files. This sample uses the C1TrueDBGrid control. |
| MultipleSelection | Select or deselect rows when you click a row. |
| SettingCellToNull | How to set the underlying datasource value to null. This sample uses the C1TrueDBGrid control. |
| ToggleGroupRows | Programmatically expand/collapse rows in a grouped grid. This sample uses the C1TrueDBGrid control. |
| TriStateCheckBox | How to add a tristate check box to the grid. This sample uses the C1TrueDBGrid control. |
| UsingC1TDBDropDown | How to use C1TrueDBDropDown to map IDs to Names. This sample uses the C1TrueDBGrid and C1TrueDBDropDown controls. |
| Zoom | Change the size of the grid. This sample uses the C1TrueDBDropDown control. |

True DBGrid for WinForms Tutorials

Twenty-Two tutorials are presented in this chapter. The tutorials assume that you are familiar with programming in Visual Studio, know what a DataSet is, and generally know how to use bound controls. The tutorials provide step-by-step instructions—no prior knowledge of **True DBGrid for WinForms** is needed. By following the steps outlined in this chapter, you will be able to create projects demonstrating a variety of **True DBGrid for WinForms** features, and get a good sense of what the **True DBGrid for WinForms** can do and how to do it.

The tutorials use an Access database, **C1NWind.mdb**. The database file **C1NWind.mdb** is in the **Common** subdirectory of the **ComponentOne Samples** folder and the tutorial projects are in the **Tutorials** folder of **Documents\ComponentOne Samples\WinForms\C1TrueDBGrid**. Although the tutorials are numbered you do not have to complete them in order; the tutorial number refers to the files in the **Tutorials** folder.

We encourage you to run the tutorial projects in Visual Studio, examine the code, and experiment with your own modifications. This is the best and quickest way to realize the full potential of **True DBGrid**. You will find that **True DBGrid** is very easy to use, and it enables you to create powerful database applications.

The tutorials assume that the database file **C1NWind.mdb** is in the **Documents\ComponentOne Samples\Common** directory, and refer to it by filename instead of the full pathname for the sake of brevity.




Note: Depending on where you store the projects and database files, you may need to change the location of the **C1NWind.mdb** reference in the DataSet.

Some of the projects created in the following tutorials are built upon later tutorials; for example the projects created in tutorials 1, 3, 6, 7, 8, and 10 are used in other tutorials. As the projects created in some of the tutorials are used again in multiple tutorials it is recommended that you save your file after completing each tutorial.


Tutorial 1: Binding True DBGrid to a DataSet

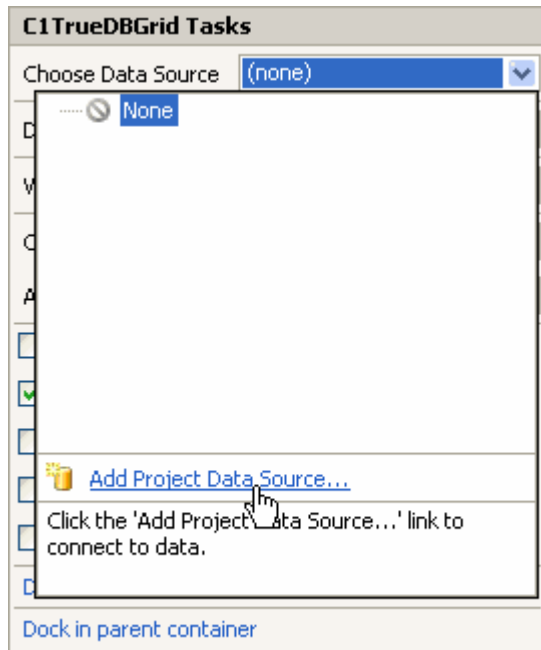
In this tutorial, you will learn how to bind **True DBGrid for WinForms** controls to a DataSet and create a fully functional database browser. You will also learn about the basic properties of the **True DBGrid for WinForms** and then be able to run the program and observe the run-time features of the grid.



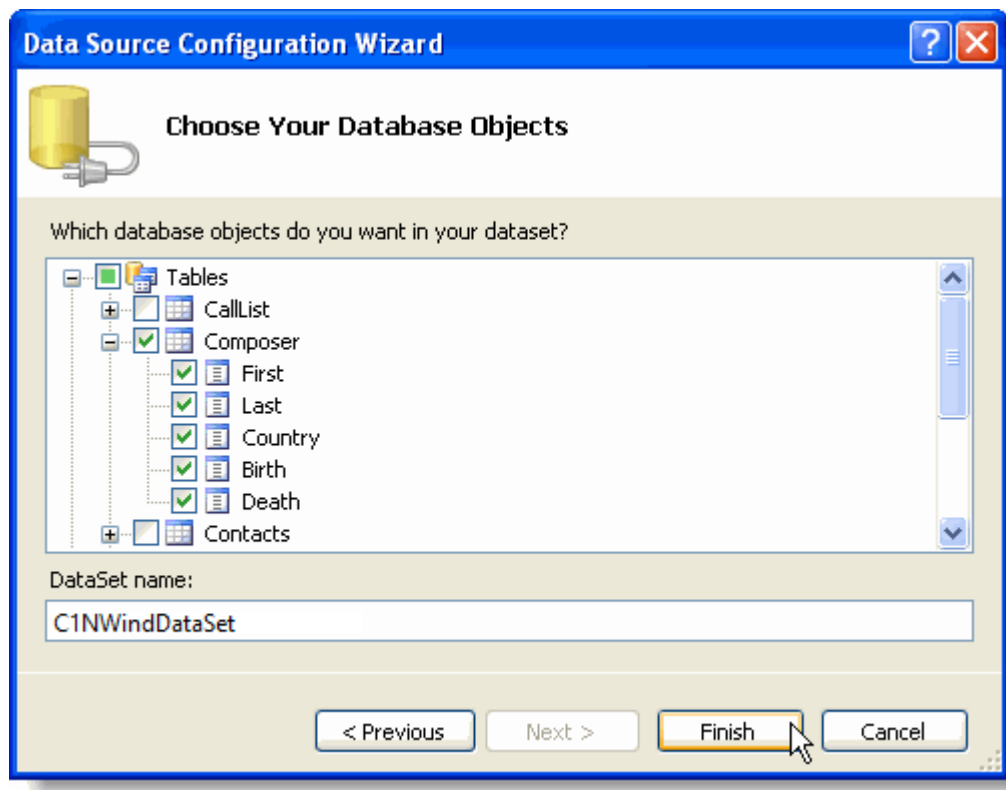
Note:  A video is available for this tutorial on the [ComponentOne Videos](#) Web page.

Complete the following steps:

1. Create a new .NET project.
2. Open the Toolbox, which is initially located on the left side of the IDE and has a hammer and a wrench as its icon. From the Toolbox, locate and double-click the **C1TrueDBGrid** icon  **C1TrueDBGrid**. The grid is added to the form and the **C1TrueDBGrid Tasks** menu appears.
3. In the menu, select the **Choose Data Source** drop-down arrow and click **Add Project Data Source**.



4. The **Data Source Configuration Wizard** appears and **Database** is selected. Click **Next**.
5. Click the **New Connection** button to locate and connect to a database.
6. Click the **Browse** button and locate the **C1NWind.mdb** file in the **Documents\ComponentOne Samples\Common** directory. Select it and click **Open**.
7. Click the **Test Connection** button to make sure that you have successfully connected to the database or server and click **OK**. The new string appears in the data connection drop-down list.
8. Click the **Next** button to continue. A dialog box will appear asking if you would like to add the data file to your project and modify the connection string. Click **No**.
9. In the next window, the **Yes, save the connection as** check box is checked by default and a name ("TDBGDemoConnectionString") has been automatically entered in the text box. Click **Next** to continue.
10. In the **Choose Your Database Objects** window, select the tables and fields that you would like in your dataset.



The DataSet is given a default name ("TDBGDemoDataSet") in the **DataSet name** text box.

11. Click **Finish** to exit the wizard. The **DataSet**, **BindingSource** and **TableAdapter** now appear on your form.
12. Resize the grid and double-click the form. Notice that Visual Studio has added the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

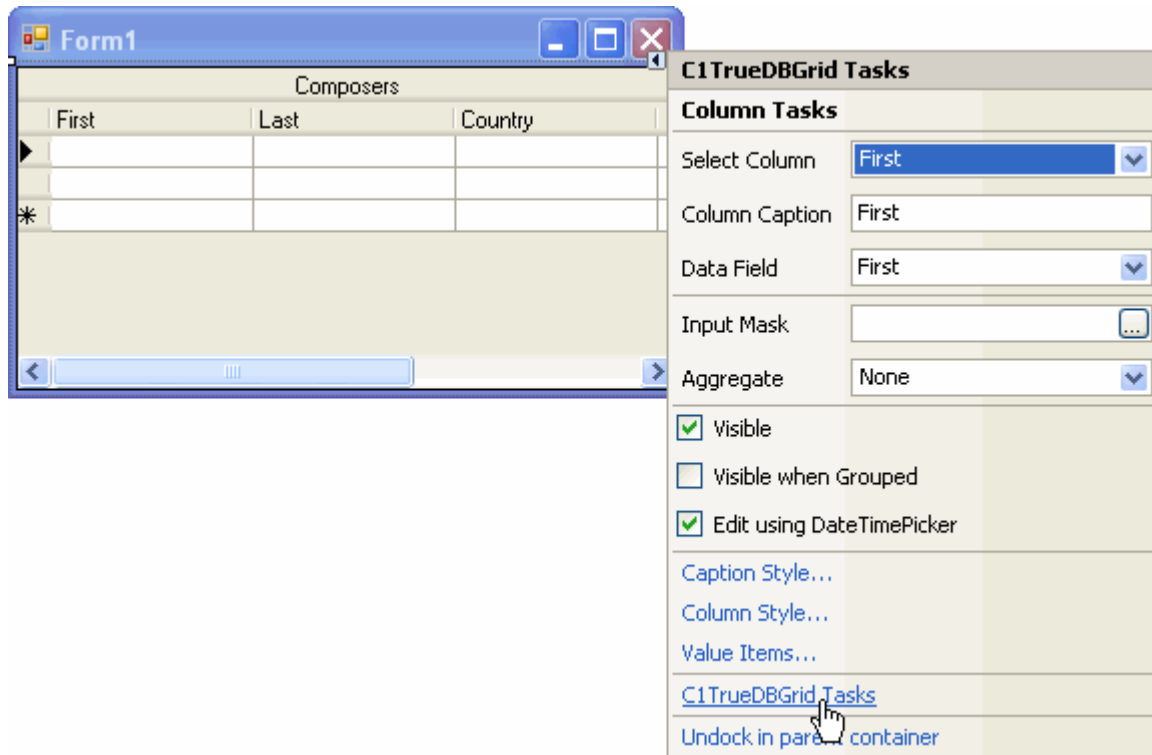
```
Me.ComposerTableAdapter.Fill(Me.DsComposer.Composer)
```

To write code in C#

C#

```
this.ComposerTableAdapter.Fill(this.DsComposer.Composer);
```

13. Click the **Design** tab to return to the designer and then select the grid.
14. Open the **C1TrueDBGrid Tasks** menu and select **C1TrueDBGrid Tasks**.



15. Check the **Enable Adding** and **Enable Deleting** check boxes. This sets the [AllowAddNew](#) and [AllowDelete](#) properties of C1TrueDBGrid1 to **True**, enabling the user to add or delete records in the grid.

Run the program and observe the following:

- **True DBGrid** retrieves the database schema information from the DataSet and automatically configures itself to display all of the fields contained in the database table. Note that the field names are used as the default column headings.
- **True DBGrid** automatically communicates with the DataSet. Any actions taken on the DataSet will be reflected in the grid.
- A fully functional database browser has been created by only writing four lines of simple code.


Refer to [Run-Time Interaction](#) and try out the instructions for navigating, editing, and configuring the grid at run time.

To end the program, close the window or press the stop button on the Visual Basic Toolbar.

Congratulations, you have successfully completed binding **True DBGrid** to a DataSet!

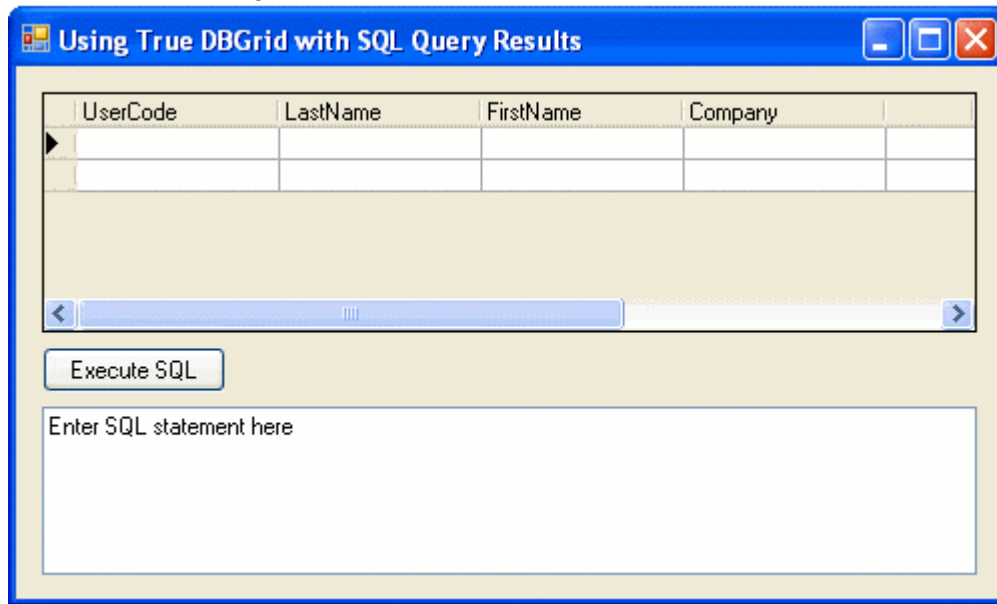
Tutorial 2: Using True DBGrid for WinForms with SQL Query Results

An important feature of **True DBGrid for WinForms** is its ability to automatically sense changes to the database at run time. In this tutorial, you will learn how to use **True DBGrid for WinForms** to display the results of ad-hoc SQL queries. In addition, it will also outline how to set up a connection to a DataSet at run time. Note that in order for the grid to automatically respond to field layout changes, you must not have defined any column properties at design time. If a layout is already defined, use the grid's **Clear Fields** context menu command to remove it. This will cause the grid to configure itself automatically at run time.

 **Note:** A video is available for this tutorial on the [ComponentOne Videos](#) Web page.

Complete the following steps:

1. Create a new .NET project.
2. Place a [C1TrueDBGrid](#) control (C1TrueDBGrid1), a **Button** (Button1), and a **TextBox** control (TextBox1) on the form. Set the **Text** property of the command button to read "Execute SQL" and set the **Text** property of the TextBox1 to "Enter SQL statement here":



3. Go to the **DataSource** property and select **Add Project Data Source** from the drop-down. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select all fields in the **Customers** table and type "DsCustomers" into the **DataSet name** box, and then finish out the wizard.
4. Visual Studio will add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.CustomersTableAdapter.Fill(Me.DsCustomers.Customers)
```

To write code in C#

C#

```
this.CustomersTableAdapter.Fill(this.DsCustomers.Customers);
```

5. Add the following code to the **Click** event of Button1:

To write code in Visual Basic

Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click  
  
    Dim sqlStr As String = TextBox1.Text  
    Dim da as OleDb.OleDbDataAdapter = New OleDb.OleDbDataAdapter (sqlStr,  
Me.CustomersTableAdapter.Connection)  
    Dim ds As DataSet = New DataSet()
```

```
ds.Clear()
Try
    da.Fill(ds, "mySQL")
    Me.ClTrueDBGrid1.DataSource = Nothing
    Me.ClTrueDBGrid1.ClearFields()
    Me.ClTrueDBGrid1.SetDataBinding(ds.Tables("mySQL"), "", False)
Catch
    MessageBox.Show("Error in SQL clause")
End Try
End Sub
```

To write code in C#

```
C#
private void button1_Click(System.Object sender, System.EventArgs e)
{
    string sqlStr = TextBox1.Text;
    da as OleDb.OleDbDataAdapter = New OleDb.OleDbDataAdapter (sqlStr,
this.CustomersTableAdapter.Connection);
    DataSet DataSet ds = new DataSet();

    ds.Clear();
    try
    {
        da.Fill(ds, "mySQL");
        this.clTrueDBGrid1.DataSource = null;
        this.clTrueDBGrid1.ClearFields();
        this.clTrueDBGrid1.SetDataBinding(ds.Tables["mySQL"], "", false);
    }
    catch ()
    {
        MessageBox.Show ("Error in SQL clause");
    }
}
```

Run the program and observe the following:

As in [Tutorial 1: Binding True DBGrid to a DataSet](#), **True DBGrid for WinForms** retrieves the database schema information from the DataSet and automatically configures itself to display the data for all fields in the database table. Note that the field names are used as the default column headings.

1. In the TextBox control, type the following SQL statement:

```
Select * from Customer
```

Press the **Execute SQL** button. The above SQL statement displays all fields from the **Customer** table and is equivalent to the default display.

2. In the TextBox control, type the following SQL statement:

```
Select Company from Customer
```

Press the **Execute SQL** button. The grid responds by displaying only one column for the *Company* field.

3. In the TextBox control, type the following SQL statement:

```
Select LastName, Company from Customer
```

Press the **Execute SQL** button. This is similar to the previous SQL statement except that two columns (*LastName* and *Company*) are now displayed.

4. In the TextBox control, type the following SQL statement:

```
Select Count(*) from Customer
```

Press the **Execute SQL** button. The above SQL statement uses an aggregate function, **Count(*)**, to return the total number of records in the **Customer** table. Even though the SQL result is not a set of records, the grid faithfully responds by displaying the number of records in a single column. By default, **Expr1000** is used as the column heading, indicating that the display is the result of an expression.

5. In the TextBox control, type the following SQL statement:

```
Select UCase(LastName) as ULAST, UCase(FirstName) AS UFIRST from Customer
```

Press the **Execute SQL** button. The above SQL statement produces two calculated columns that display the *LastName* and *FirstName* fields in upper case. The grid also displays the (assigned) calculated column names, *ULAST* and *UFIRST*, as the column headings.

6. In the TextBox control, type the following SQL statement:

```
SELECT * FROM Customer WHERE FirstName = "Jerry"
```

Press the **Execute SQL** button. The above SQL statement displays only records with *FirstName* equal to **Jerry**.

7. In the TextBox control, type the following SQL statement:

```
SELECT * FROM Customer ORDER BY LastName
```

Press the **Execute SQL** button. The above SQL statement displays records in alphabetical order according to the *LastName* field.

You can also use an SQL statement to join two database tables, as demonstrated in [Tutorial 3: Linking Multiple True DBGrid Controls](#).

This concludes tutorial 2; you've successfully completed using **True DBGrid** with SQL query results.

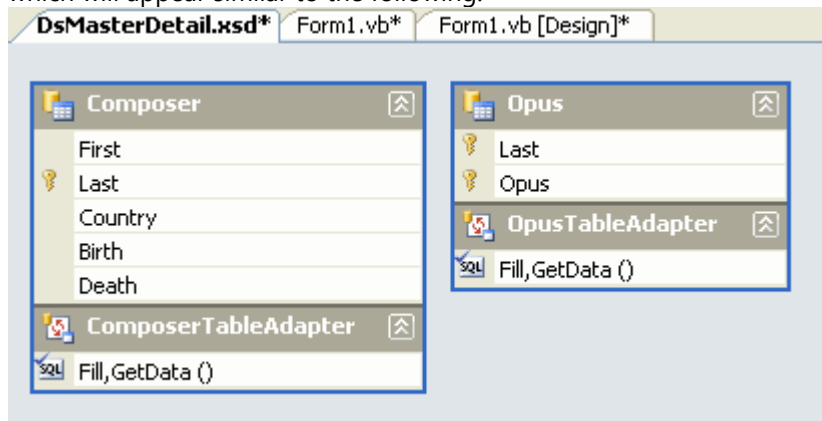
Tutorial 3: Linking Multiple True DBGrid Controls

This tutorial demonstrates how to link multiple **True DBGrid for WinForms** controls using a Master Detail dataset.

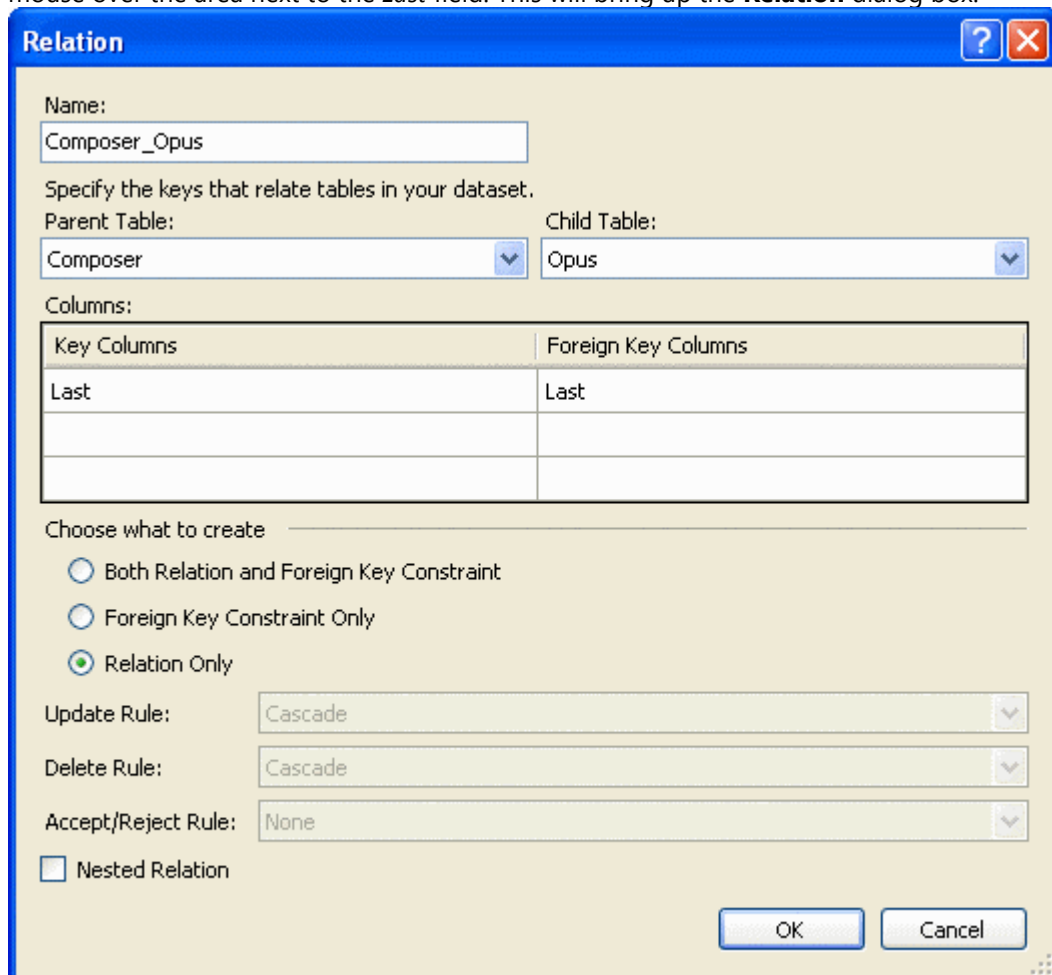
Complete the following steps:

1. Create a new .NET project.
2. Navigate to the Visual Studio Toolbox and double-click the **C1TrueDBGrid** item twice to add two **C1TrueDBGrid** controls to the form (**C1TrueDBGrid1** and **C1TrueDBGrid2**).
3. In the **C1TrueDBGrid1** control's **C1TrueDBGrid Tasks** menu, locate the **Choose Data Source** drop-down and select **Add Project Data Source**.
4. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select all fields in the **Composer** table and all fields in the **Opus** table, and type "DsMasterDetail" into the **DataSet name** box, and then finish out the wizard.

5. Double-click **DsMasterDetail.xsd** in the Solution Explorer window. This will open the DsMasterDetail.xds file, which will appear similar to the following:

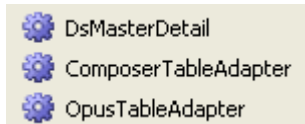


6. To make a relation between two tables, click and hold down the mouse button in the area next to the *Last* field in **Composer**, and then drag the mouse up over the **Composers** table over to the **Opus** table, then release the mouse over the area next to the *Last* field. This will bring up the **Relation** dialog box:

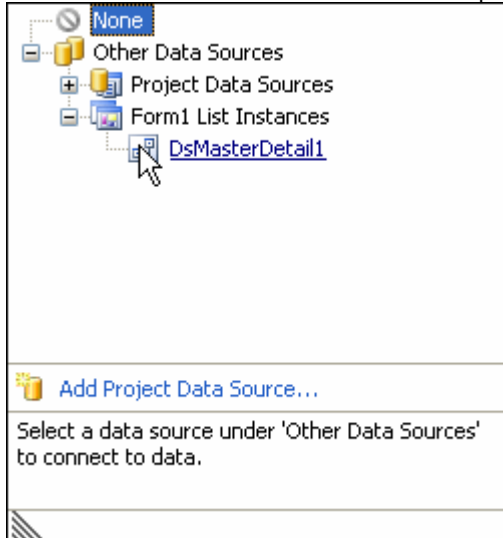


Make sure **Parent Table** is set to **Composer** and the **Child Table** is set to **Opus**. In addition make sure both fields are set to the *Last* column and that the **Relation Only** is selected (as in the preceding screenshot). Click **OK** and exit the **Edit Relation** dialog box.

7. Now go to the **Build** menu of Visual Studio and choose **Build Solution**. This will ensure that this new relation is available in the project.
8. Return to the form's Design view and in the Toolbox, locate the **<Your Project Name> Components** tab. Add an instance of the **DsMasterDetail**, **ComposerTable**, and **OpusTableAdapter** to the form.

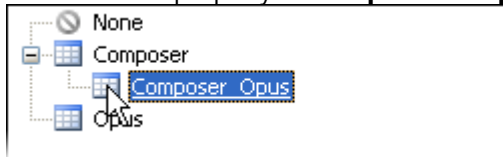


9. Now in the Properties window, set the **DataSource** property for the first **C1TrueDBGrid** control to **DsMasterDetail1** and the **DataMember** property to **Composer**.



If prompted to replace the column layout, click **Yes**.

10. For the second **C1TrueDBGrid** control, set the **DataSource** property to **DsMasterDetail1** and the **DataMember** property to **Composer.Composer_Opus**.



If prompted to replace the column layout, click **Yes**.

11. All that is left is to populate the **DataAdapters**.
12. Double-click the form to switch to Code view and create the **Form_Load** event handler. Add the following code to the **Load** event of Form1:

To write code in Visual Basic

Visual Basic

```
Me.ComposerTableAdapter1.Fill(Me.DsMasterDetail1.Composer)
Me.OpusTableAdapter1.Fill(Me.DsMasterDetail1.Opus)
```

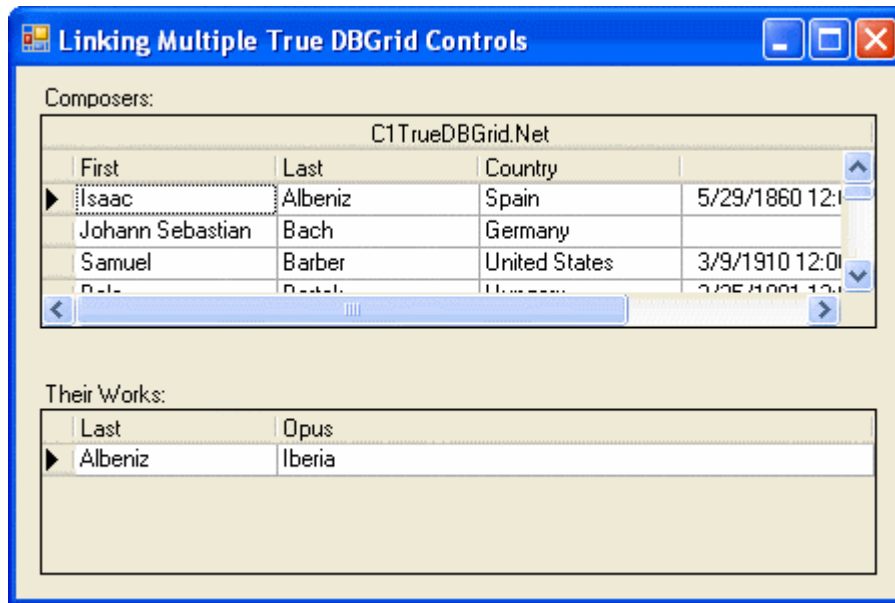
To write code in C#

C#

```
this.composerTableAdapter1.Fill(this.dsMasterDetail1.Composer);
this.opusTableAdapter1.Fill(this.dsMasterDetail1.Opus);
```

Run the program and observe the following:

- When Form1 is loaded, C1TrueDBGrid1 and C1TrueDBGrid2 retrieve the database schema information from DsMasterDetail:



- Change the current record position of the first grid by clicking on different rows. Observe that C1TrueDBGrid2 (the *detail* grid) will configure itself to display a new record set every time the row changes in C1TrueDBGrid1 (the *master* grid).

This concludes this tutorial; you've successfully completed linking multiple **True DBGrid** controls.

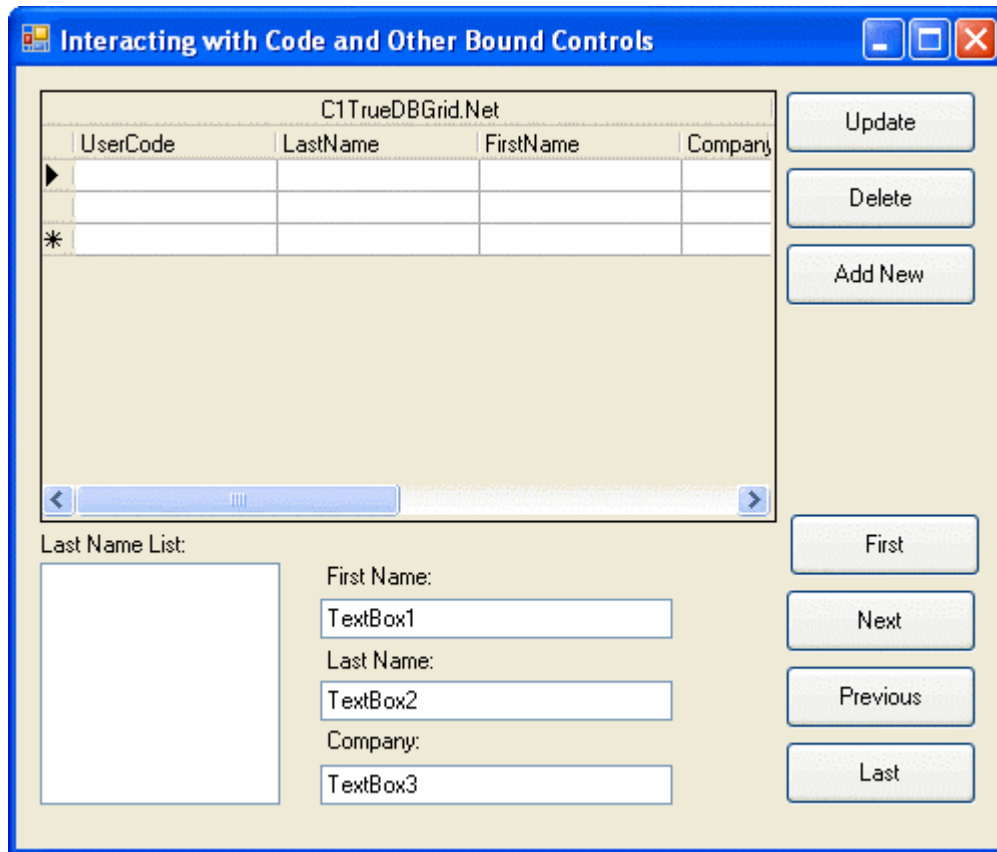
Tutorial 4: Interacting with Code and Other Bound Controls

In this tutorial, you will learn how **True DBGrid** interacts with other bound controls and with code that manipulates the same DataSet to which the grid is bound.

Complete the following steps:

1. Create a new .NET project.
2. Place the following controls on the form (Form1) as shown in the figure below:
 - C1TrueDBGrid control (C1TrueDBGrid1)
 - ListBox control (ListBox1)
 - Three text controls (TextBox1 to 3)
 - Seven buttons (Named btnFirst, btnNext, btnPrevious, btnLast, btnDelete, btnUpdate, btnAdd)
 - Four labels (Label1 to 4)

Set the **Text** properties for each of the labels and buttons seen below:



3. In the **C1TrueDBGrid Tasks** menu, locate the **Choose Data Source** drop-down and select **Add Project Data Source**. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select all fields in the **Customers** table and type "DsCustomers" into the **DataSet name** box, and then finish out the wizard.
4. Visual Studio adds the following code to the **Form_Load** event :

To write code in Visual Basic

Visual Basic

```
Me.CustomersTableAdapter.Fill(Me.DsCustomers.Customers)
```

To write code in C#

C#

```
this.CustomersTableAdapter.Fill(this.DsCustomers.Customers);
```

5. Now for each of the four Buttons (Button4, 5, 6, 7) on the lower right in the above image, add the following code:

To write code in Visual Basic

Visual Basic

```
Private Sub btnFirst_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnFirst.Click
```

```
    ' Move to the first record in the grid.
```

```
Me.ClTrueDBGrid1.MoveFirst()  
End Sub  
  
Private Sub btnNext_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles btnNext.Click  
  
    ' Move to the next record in the grid.  
    Me.ClTrueDBGrid1.MoveNext()  
End Sub  
  
Private Sub btnPrevious_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles btnPrevious.Click  
  
    ' Move to the previous record in the grid.  
    Me.ClTrueDBGrid1.MovePrevious()  
End Sub  
  
Private Sub btnLast_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles btnLast.Click  
  
    ' Move to the last record in the grid.  
    Me.ClTrueDBGrid1.MoveLast()  
End Sub
```

To write code in C#

```
C#  
  
private void btnFirst_Click(System.object sender, System.EventArgs e)  
{  
    // Move to the first record in the grid.  
    this.clTrueDBGrid1.MoveFirst();  
}  
  
private void btnNext_Click(System.object sender, System.EventArgs e)  
{  
    // Move to the next record in the grid.  
    this.clTrueDBGrid1.MoveNext();  
}  
  
private void btnPrevious_Click(System.object sender, System.EventArgs e)  
{  
    // Move to the previous record in the grid.  
    this.clTrueDBGrid1.MovePrevious();  
}  
  
private void btnLast_Click(System.object sender, System.EventArgs e)  
{  
    // Move to the last record in the grid.  
    this.clTrueDBGrid1.MoveLast();  
}
```

6. Set the code for the three buttons on the upper right in the above image:

To write code in Visual Basic**Visual Basic**

```
Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnDelete.Click

    ' Delete the record and save the changes to the database.
    Me.ClTrueDBGrid1.Delete()
    Call UpdateCustomers()
End Sub

Private Sub BtnUpdate_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnUpdate.Click

    ' Update the grid, call UpdateCustomers to save.
    Me.ClTrueDBGrid1.UpdateData()
    Call UpdateCustomers()
End Sub

Private Sub BtnAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAdd.Click

    ' This "Add New" button moves the cursor to the "new (blank) row" at the end
    so that user can start adding data to the new record.
    ' Move to last record, "new row" will be visible.
    Me.ClTrueDBGrid1.MoveLast()

    ' Move to the "addnew row", and set focus to the grid.
    Me.ClTrueDBGrid1.Row = Me.ClTrueDBGrid1.Row + 1
    Me.ClTrueDBGrid1.Select()
End Sub
```

To write code in C#**C#**

```
private void btnDelete_Click(System.object sender, System.EventArgs e)
{
    // Delete the record and save the changes to the database.
    this.clTrueDBGrid1.Delete();
    UpdateCustomers();
}

private void BtnUpdate_Click(System.object sender, System.EventArgs e)
{
    // Update the grid, call UpdateCustomers to save.
    this.clTrueDBGrid1.UpdateData();
    UpdateCustomers();
}

private void BtnAdd_Click(System.object sender, System.EventArgs e)
```

```
{  
    // This "Add new" button moves the cursor to the "new (blank) row" at the  
end so that user can start adding data to the new record.  
    // Move to last record, "new row" will be visible.  
    this.clTrueDBGrid1.MoveLast();  
  
    // Move to the "addnew row", and set focus to the grid.  
    this.clTrueDBGrid1.Row = this.clTrueDBGrid1.Row + 1;  
    this.clTrueDBGrid1.Select();  
}
```

7. Now add the UpdateCustomers subroutine. It sends information back to the DataSet from temporary DataTables:

To write code in Visual Basic

Visual Basic

```
Private Sub UpdateCustomers()  
    Me.CustomersTableAdapter.Connection.Open()  
    Dim UpdatedRows As System.Data.DataSet  
    Dim InsertedRows As System.Data.DataSet  
    Dim DeletedRows As System.Data.DataSet  
  
    ' These Data Tables hold any changes that have been made to the dataset  
since the last update.  
    UpdatedRows = Me.DsCustomers.GetChanges(DataRowState.Modified)  
    InsertedRows = Me.DsCustomers.GetChanges(DataRowState.Added)  
    DeletedRows = Me.DsCustomers.GetChanges(DataRowState.Deleted)  
  
    Try  
  
        ' For each of these, we have to make sure that the Data Tables contain any  
records, otherwise, we will get an error.  
        If Not UpdatedRows Is Nothing Then  
Me.CustomersTableAdapter.Update(UpdatedRows)  
        If Not InsertedRows Is Nothing Then  
Me.CustomersTableAdapter.Update(InsertedRows)  
        If Not DeletedRows Is Nothing Then  
Me.CustomersTableAdapter.Update(DeletedRows)  
        Catch eUpdate As System.Exception  
            MessageBox.Show ("Caught exception: " & eUpdate.Message)  
        End Try  
  
        Me.CustomersTableAdapter.Connection.Close()  
    End Sub
```

To write code in C#

C#

```
private void UpdateCustomers()  
{  
    this.CustomersTableAdapter.Connection.Open();  
}
```

```
System.Data.DataSet UpdatedRows;
System.Data.DataSet InsertedRows;
System.Data.DataSet DeletedRows;

// These Data Tables hold any changes that have been made to the dataset
since the last update.
UpdatedRows = this.DsCustomers.GetChanges(DataRowState.Modified);
InsertedRows = this.DsCustomers.GetChanges(DataRowState.Added);
DeletedRows = this.DsCustomers.GetChanges(DataRowState.Deleted);

try
{
    // For each of these, we have to make sure that the Data Tables contain
    any records, otherwise, we will get an error.
    if (! UpdatedRows == null )
        this.CustomersTableAdapter.Update(UpdatedRows)
    if (! InsertedRows == null )
        this.CustomersTableAdapter.Update(InsertedRows)
    if (! DeletedRows == null )
        this.CustomersTableAdapter.Update(DeletedRows)
}
catch (System.Exception eUpdate)
{
    MessageBox.Show ("Caught exception: " + eUpdate.Message);
}
this.CustomersTableAdapter.Connection.Close();
}
```

8. Now back in the .NET designer, click the **ListBox1** control. In the Properties window set its **DataSource** property to **CustomersBindingSource**, and its **DisplayMember** property to **LastName**.
9. Click on the first TextBox (TextBox1) on the form. In the Properties window, expand its **DataBindings** object. Set the **Text** property under the **DataBindings** object to **CustomersBindingSource - FirstName**. In a similar manner set the same **Text** property under the associated **DataBindings** object for the next two TextBoxes. TextBox2 should be set to **LastName**, and TextBox3 should be set to **Company**.
10. Finally in the Properties window set the **AllowAddNew**, **AllowDelete**, and **AllowUpdate** properties to **True** for the **C1TrueDBGrid** control.

Run the program and observe the following:


- Use the mouse or the keyboard to change the current row position in the grid, and observe the other bound controls (ListBox1 and the TextBoxes) changing their record positions along with the grid, even though they contain no code.
- Click the **Next**, **Previous**, **Last**, and **First** buttons and observe that they have the same effects as the corresponding buttons on the Data control.
- Modify data in a few cells (in the same row) on the grid. Press the **Update** button. Observe that the modified data has been updated to the database and the pencil icon on the record selector column disappears. Other bound controls on the form now display the modified data.
- Modify data in one or more of the Text controls. Press the **Update** or the **Next** button. The grid will automatically update its data to reflect the new changes.
- Move the current cell of the grid to any record you wish to delete, then click the **Delete** button. The record will be deleted and disappears from the grid. The grid automatically moves the current row to the record after the deleted record. Other bound controls on the form also respond by moving their record positions.

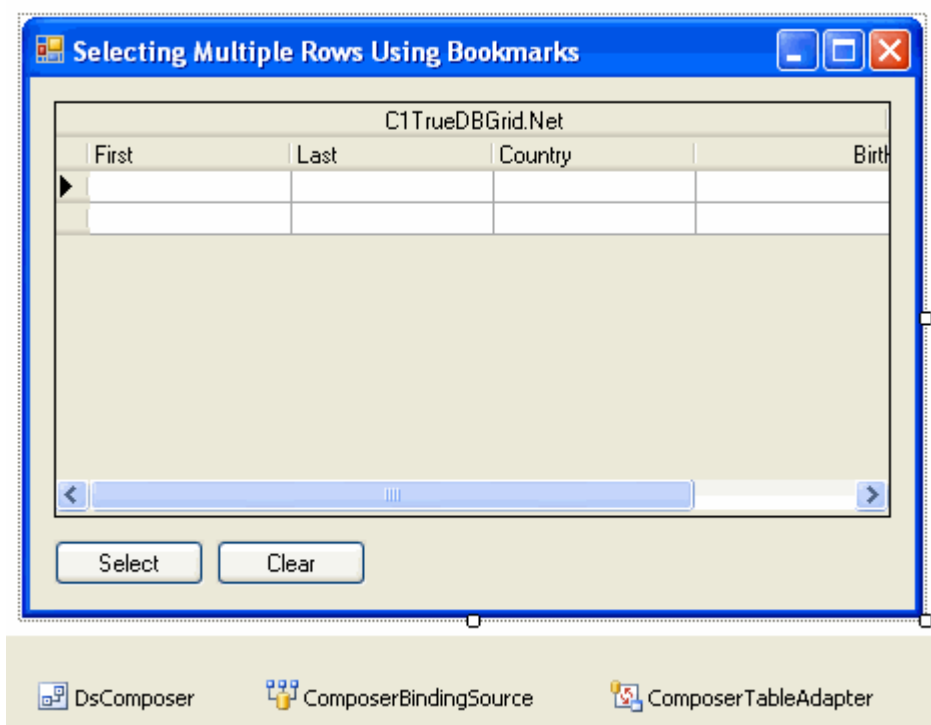
This concludes this tutorial; you've successfully completed interacting with code and other bound controls.

Tutorial 5: Selecting Multiple Rows Using Bookmarks

In this tutorial, you will learn how to select and highlight records that satisfy specified criteria. A group of similar items is generally implemented as a collection in **True DBGrid**. When manipulating a group of items in **True DBGrid**, use techniques similar to those described here.

Complete the following steps:

1. Create a new .NET project.
2. From the Toolbox on the left side of the IDE add two command buttons and a **C1TrueDBGrid** control onto the form. The **C1TrueDBGrid** icon looks like this:
 C1TrueDBGrid
3. Set Button1's **Text** property to "**Select**" and set Button2's **Text** property to "**Clear**."
4. In the **C1TrueDBGrid Tasks** menu, locate the **Choose Data Source** drop-down and select **Add Project Data Source**. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select all fields in the **Composers** table and type "DsComposers" into the **DataSet name** box, and then finish out the wizard.



5. Visual Studio adds the following code to the **Form_Load** event :

To write code in Visual Basic

Visual Basic

```
Me.ComposerTableAdapter.Fill(Me.DsComposer.Composer)
```

To write code in C#

C#

```
this.ComposerTableAdapter.Fill(this.DsComposer.Composer);
```


6. We can easily select and deselect rows in **True DBGrid** by manipulating the [SelectedRowCollection](#). To select rows, add the following code to the **Click** event of Button1:

To write code in Visual Basic

Visual Basic

```
Dim l As Integer
For l = 0 To Me.DsComposer.Composer.Rows.Count - 1
    If Me.DsComposer.Composer.Rows(l).Item("Country") = "Germany" Then
        Me.C1TrueDBGrid1.SelectedRows.Add(l)
    End If
Next
Me.C1TrueDBGrid1.Refresh()
```

To write code in C#

C#

```
int l;
for (l = 0 ; l < this.DsComposer.Composer.Rows.Count; l++)
{
    if (this.DsComposer.Composer.Rows[l].["Country"] == "Germany")
    {
        this.c1TrueDBGrid1.SelectedRows.Add(l);
    }
}
this.c1TrueDBGrid1.Refresh();
```

7. To deselect rows, add the following code to the **Click** event of Button2:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.SelectedRows.Clear()
```

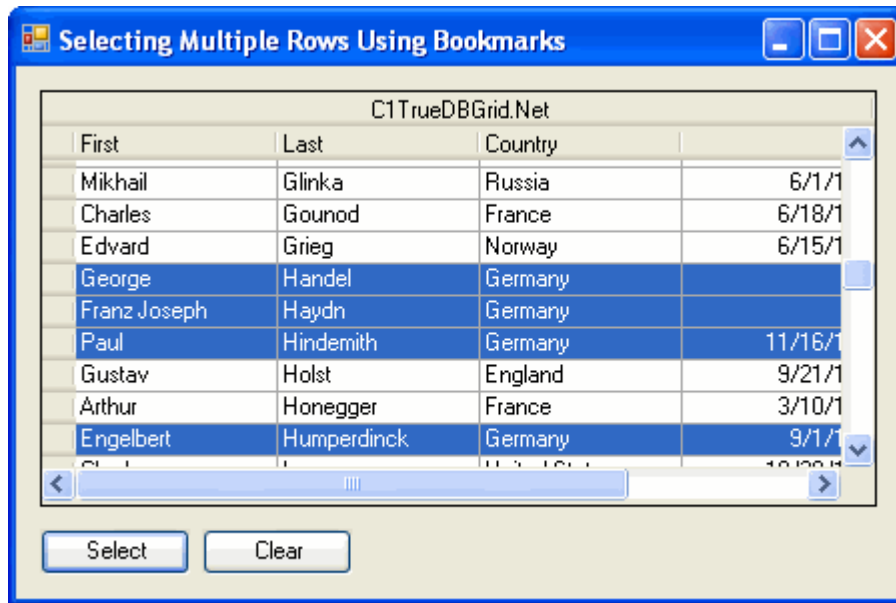
To write code in C#

C#

```
this.c1TrueDBGrid1.SelectedRows.Clear();
```

Run the program and observe the following:

- C1TrueDBGrid1 retrieves the database schema information from the DataSet and automatically configures itself to display all of the fields in the joined database tables. This is again similar to the behavior of the grid in Tutorial 1.
- Click the **Select** button and observe that all records with the *Country* field equal to **Germany** will be highlighted.




- To deselect the highlighted records, click the **Clear** button. Alternatively, clicking anywhere on a grid cell will also clear the selected rows.

This concludes this tutorial; you've successfully completed selecting multiple rows using bookmarks.

Tutorial 6: Defining Unbound Columns in a Bound Grid

In this tutorial, you will learn how to use the [UnboundColumnFetch](#) event to display two fields (*FirstName* and *LastName*) together in one column. You will also learn how to use an SQL statement to create a join between two tables in a database.

Complete the following steps:

1. Create a new .NET project.
2. From the Toolbox on the left side of the IDE double-click the [C1TrueDBGrid](#) icon to add the control to the form. The [C1TrueDBGrid](#) icon looks like this:
 C1TrueDBGrid
3. In the **C1TrueDBGrid Tasks** menu, locate the **Choose Data Source** drop-down and select **Add Project Data Source**. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select all fields in the **Contacts** table and type "DsContacts" into the **DataSet name** box, and then finish out the wizard.
4. Double click **DsContacts.xsd** in the Solution Explorer window to edit it in the Designer. Right click on the **Contacts** table and choose **Configure** from the context menu.
5. Modify the SQL string in the **Table Adapter Configuration Wizard** to:

```
SELECT Customers.FirstName, Customers.LastName, Customers.CustType,
Contacts.ContactType, Contacts.Callback, Contacts.ContactDate, Contacts.UserCode,
Customers.UserCode AS Expr1 FROM Contacts INNER JOIN Customers ON
Contacts.UserCode = Customers.UserCode
```
6. The **Contacts** table is now joined with the **Customers** table. Click the **Finish** button to exit the wizard.
7. Return to Design view and if prompted to replace existing column layout, click **Yes**.
Note: If all of the columns are not showing up in [C1TrueDBGrid](#), select the DataSource again from the drop-down box in the **C1TrueDBGrid Tasks** menu. Here, you would re-select the **Contacts** table under **DsContacts**.
8. Declare a new global DataTable object in Form1:

To write code in Visual Basic

Visual Basic

```
Dim dtCopy As New DataTable
```

To write code in C#

C#

```
DataTable dtCopy = new DataTable;
```

9. Now in the **Form_Load** event add the following code. The first line, supplied by Visual Studio, fills the dataset and the second line makes a copy of this DataSet, which we will use later to populate the unbound column:

To write code in Visual Basic

Visual Basic

```
Me.ContactsTableAdapter.Fill(Me.DsContacts.Contacts)
dtCopy = Me.DsContacts.Tables(0).Copy()
```

To write code in C#

C#

```
this.ContactsTableAdapter.Fill(this.DsContacts.Contacts);
dtCopy = this.DsContacts.Tables(0).Copy();
```

10. To create an unbound column, open up the **C1TrueDBGrid Designer** by clicking on the **ellipsis** button (...) next to the **Columns** property in the Properties window. Next click the **Appendcolumn** button to create a new column. Set the new column's **Caption** property to "Name" in the left pane. Notice that a value resides in the *Caption* field, but no value in the *DataField*, which is how the grid knows that this is an unbound column. The grid now knows to fire the [UnboundColumnFetch](#) event. Click the **OK** button to close the **C1TrueDBGrid Designer**.
11. Open the [SplitCollection](#) editor by clicking on the **ellipsis** button next to the **Splits** property in the Properties window. Now open up the [C1DisplayColumnCollection](#) editor by clicking on the **ellipsis** button next to the **DisplayColumns** property. In this editor, find the unbound column in the left pane that we just created. It is positioned as the last column in the grid. The DisplayColumns Collection determines the position of the field. Maneuver the column to the desired location by using the up and down arrow buttons in the left pane. Then in the right pane, set its **Visible** property equal to **True**. Now our unbound column is visible to the end-user and not just the **True DBGrid for WinForms** control.
- You can hide columns here that are used in the unbound column. Select the *FirstName* column from the left pane, then in the right, set its **Visible** property equal to **False**. This hides the *FirstName* column from view. Repeat, selecting the *LastName* column.
- Select **OK** to close the [C1DisplayColumnCollection](#) editor and click **OK** again to close the [SplitCollection](#) editor.
12. Add the following code to the **UnboundColumnFetch** event. This code uses dtCopy to gather values to place into the unbound column, then setting these values equal to e.Value, places the value into the unbound column:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_UnboundColumnFetch(ByVal sender As System.Object,
ByVal e As C1.Win.C1TrueDBGrid.UnboundColumnFetchEventArgs) Handles
C1TrueDBGrid1.UnboundColumnFetch
    If e.Column.Caption = "Name" AndAlso e.Row < dtCopy.Rows.Count Then
        e.Value = Me.C1TrueDBGrid1(e.Row, "FirstName").ToString + " " +
```

```
Me.C1TrueDBGrid1(e.Row, "LastName").ToString
    End If
End Sub
```

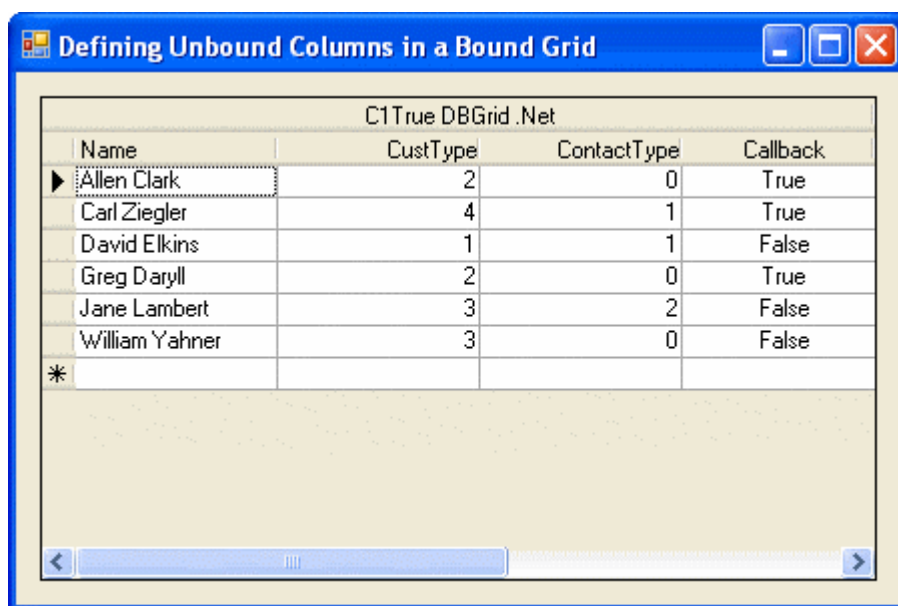
To write code in C#

```
C#

private void c1TrueDBGrid1_UnboundColumnFetch(object sender,
C1.Win.C1TrueDBGrid.UnboundColumnFetchEventArgs e)
{
    if(e.Column.Caption == "Name" && e.Row < dtCopy.Rows.Count)
    {
        e.Value = this.c1TrueDBGrid1[e.Row, "FirstName"].ToString() + " " +
this.c1TrueDBGrid1[e.Row, "LastName"].ToString();
    }
}
```

Run the program and observe the following:

When the application runs, it should look like the following:



- C1TrueDBGrid1 displays data from the joined table according to the five columns configured at design time.
- The first column displays the combined *FirstName* and *LastName* fields as defined in the [UnboundColumnFetch](#) event.
- The *CustType*, *ContactType* and *Callback* columns display numeric values that are quite cryptic to users and provide an unappealing data presentation. In the next three tutorials (7, 8, and 9), techniques will be illustrated that improve both the display and the user interface.

This concludes this tutorial; you've successfully completed defining unbound columns in a bound grid.

Tutorial 7: Displaying Translated Data with the Built-In Combo

In this tutorial, you will learn how to use the [ValueItems](#) object to display translated text and enable the grid's built-in drop-down combo for editing—all without writing a single line of code.

Complete the following steps:

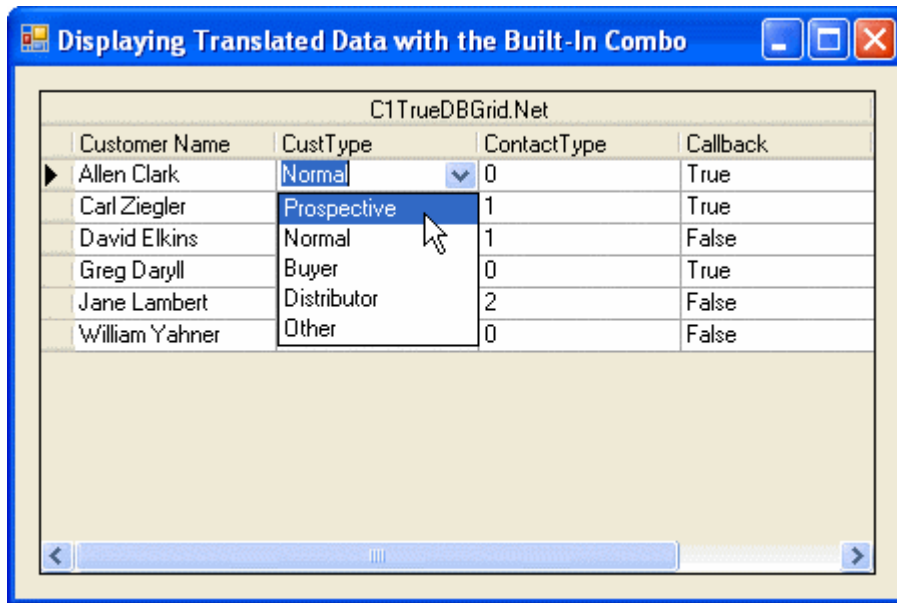
1. Start with the project created in [Tutorial 6: Defining Unbound Columns in a Bound Grid](#).
2. Make sure the [C1TrueDBGrid](#) has focus, then click the **ellipsis** button next to the **Columns** property in the Properties window. This brings up the **C1TrueDBGrid Designer**.
3. Select the *CustType* column member. Then in the left pane, click the expand icon next to the **ValueItems** property. This will display all of the members of the Valueitems object.
4. Click on the **ellipsis** button next to the **Values** property in the **C1TrueDBGrid Designer**. This brings up the **ValueItemCollection Editor**.
5. In the left pane create five new **ValueItem** objects by clicking on the **Add** button five times. Notice that a **ValueItem** has two properties: **DisplayValue** and **Value**.
6. Enter the following DisplayValue/Value pairs into the right pane, then close the **ValueItemCollection Editor**:

| DisplayValue | Value |
|--------------|-------|
| Value | 0 |
| Prospective | 1 |
| Normal | 2 |
| Buyer | 3 |
| Distributor | 4 |
| Other | 5 |

7. Under the Valueitems object in the **C1DataColumn Editor**, set the [Presentation](#) property to **ComboBox**, and the [Translate](#) property to **True**.
8. Click the **OK** button at the bottom of the **Property Pages** dialog box to accept the changes.

Run the program and observe the following:

- C1TrueDBGrid1 displays data from the joined tables as in [Tutorial 6: Defining Unbound Columns in a Bound Grid](#).
- The *CustType* column now displays the translated text instead of numeric values.
- Click a cell in the *CustType* column to make it the current cell. Notice that a drop-down button appears at the right edge of the cell.
- Click the drop-down button or press ALT+DOWN ARROW to display the built-in combo box containing translated values, as shown in the following figure. Change the data in the current cell by selecting the desired item from the combo box.




This concludes this tutorial; you've successfully completed displaying translated data with the built-in combo.

Tutorial 8: Attaching a Drop-Down Control to a Grid Cell

In this tutorial, you will learn how to attach a multicolumn **True DBDropDown** control to a grid cell. Unlike the built-in combo demonstrated in [Tutorial 7: Displaying Translated Data with the Built-In Combo](#), the **C1TrueDBDropDown** control can be bound to a data source, which makes it ideal for data entry involving a secondary lookup table. The drop-down control appears whenever the user clicks a button within the current cell. This button appears automatically when the user gives focus to a column that has a drop-down control connected to it.

Complete the following steps:

1. Start with the project constructed in [Tutorial 6: Defining Unbound Columns in a Bound Grid](#).
2. Add a **C1TrueDBDropDown** control (C1TrueDBDropDown1) to the form. The icon for the **C1TrueDBDropDown** looks like the following:
 C1TrueDBDropDown
3. Go to the **DataSource** property and select **Add Project Data Source** from the drop-down. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select the *TypeID* and *TypeDesc* fields in the **CustType** table and type "DsCustType" into the **DataSet name** box, and then finish out the wizard.
4. Visual Studio adds the following code to the **Load** event of Form1 to fill the new dataset:

To write code in Visual Basic

Visual Basic

```
Me.CustTypeTableAdapter.Fill(Me.DsCustType.CustType)
```

To write code in C#

C#

```
this.CustTypeTableAdapter.Fill(this.DsCustType.CustType);
```

5. Then again in the **Load** event of Form1, add the following code to set the C1TrueDBDropDown1 to the *CustType* column:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns("CustType").DropDown = Me.C1TrueDBDropDown1
```

To write code in C#

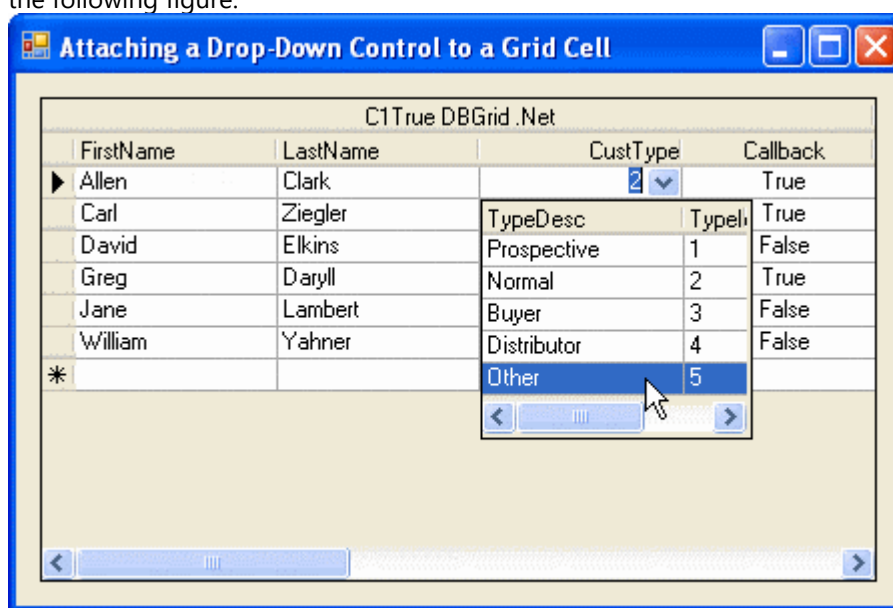
C#

```
this.c1TrueDBGrid1.Columns("CustType").DropDown = this.c1TrueDBDropDown1;
```

6. In the Properties window set the [DisplayMember](#) property of the C1TrueDBDropDown1 to **TypeID**. This property informs the drop-down which column will be synchronized with the grid column that the drop-down is bound to.

Run the program and observe the following:

- C1TrueDBGrid1 displays data from the joined table as in [Tutorial 6: Defining Unbound Columns in a Bound Grid](#).
- Click a cell in the *CustType* column to make it the current cell as indicated by the highlight. A button will be displayed at the right edge of the cell. Click the button to display the **True DBDropDown** control as shown in the following figure.



- Use the UP ARROW and DOWN ARROW keys to move the highlight bar of C1TrueDBDropDown control. If another cell in the grid is clicked, C1TrueDBDropDown will lose focus and become invisible.
- Select any item in the C1TrueDBDropDown. The current cell in the grid will be updated with the selected item, and the C1TrueDBDropDown will disappear until editing is initiated again.
- If the current cell is moved to another column, the button will disappear from the cell in the *CustType* column.

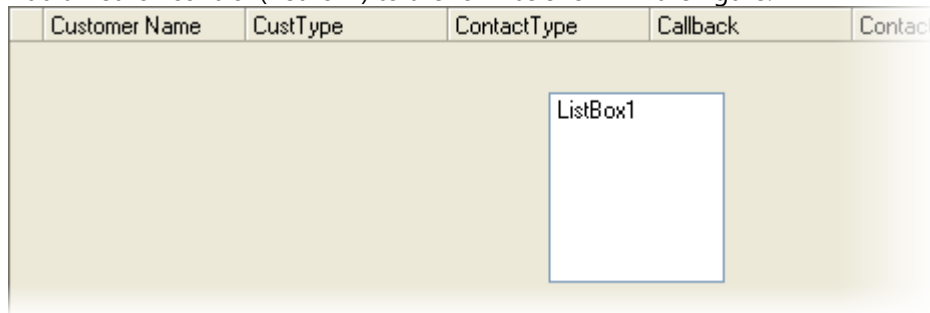
You've successfully completed attaching a drop-down control to a grid cell; this concludes tutorial 8

Tutorial 9: Attaching an Arbitrary Drop-Down Control to a Grid Cell

In this tutorial, you will learn how to drop-down an arbitrary control from a grid cell. This example uses a `ListBox` control containing pre-defined input values in order to facilitate user data entry. The list will drop down whenever the user initiates editing, such as by clicking the current cell. You will also learn how to place a button in the cell which, when clicked, will cause the `ListBox` control to appear. You can drop-down any control from a grid cell using techniques similar to those described in this tutorial.

Complete the following steps:

1. Start with the project constructed in [Tutorial 6: Defining Unbound Columns in a Bound Grid](#).
2. Add a `ListBox` control (`ListBox1`) to the form as shown in the figure.



3. Set the **Visible** property of `ListBox1` to **False**.

Adding code to drop down a `ListBox` control

The `CustType` field in the second column (Column1) of the grid displays numeric values ranging from 1 through 5, which represent the following customer types:

- 1 = Prospective
- 2 = Normal
- 3 = Buyer
- 4 = Distributor
- 5 = Other

Drop down `ListBox1`, which will contain textual customer type descriptions, and allow users to double-click an item in order to enter the associated value into the grid.

1. Include the following code in the general declaration section of `Form1`. Adding these namespaces will simplify the code we will need to write later.

To write code in Visual Basic

Visual Basic

```
Imports System.Data
Imports System.Data.OleDb
Imports System.IO
Imports System.Collections
```

To write code in C#

C#

```
using System.Data;
using System.Data.OleDb;
using System.IO;
using System.Collections;
```


2. In the **Load** event of Form1, add the code to include the customer types to ListBox1. Also, place a button in the *CustType* column using the [Button](#) property. The **Form1_Load** event handler now looks like this:

To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Me.ContactsTableAdapter.Fill(Me.DsContacts.Contacts)

    ' Add customer types to ListBox1.
    With Me.ListBox1
        .Items.Add("Prospective")
        .Items.Add("Normal")
        .Items.Add("Buyer")
        .Items.Add("Distributor")
        .Items.Add("Other")
        .Visible = False
    End With

    ' Place a button in the CustType column.
    Me.ClTrueDBGrid1.Splits(0).DisplayColumns("CustType").Button = True
End Sub
```

To write code in C#

C#

```
private void Form1_Load(System.Object sender, System.EventArgs e)
{
    this.ContactsTableAdapter.Fill(this.DsContacts.Contacts);

    // Add customer types to ListBox1.
    this.listBox1.Items.Add("Prospective");
    this.listBox1.Items.Add("Normal");
    this.listBox1.Items.Add("Buyer");
    this.listBox1.Items.Add("Distributor");
    this.listBox1.Items.Add("Other");
    this.listBox1.Visible = false;

    // Place a button in the CustType column.
    this.clTrueDBGrid1.Splits[0].DisplayColumns["CustType"].Button = true;
}
```

3. If a cell in the *CustType* column becomes current, a button will be placed at the right edge of the cell. Clicking the button will trigger the grid's [ButtonClick](#) event. We will drop down ListBox1 whenever the button is clicked:

To write code in Visual Basic

Visual Basic

```
Private Sub ClTrueDBGrid1_ButtonClick(ByVal sender As Object, ByVal e As
Cl.Win.ClTrueDBGrid.ColEventArgs) Handles ClTrueDBGrid1.ButtonClick
    With ListBox1
```

```

        .Left = Me.C1TrueDBGrid1.Left + Me.C1TrueDBGrid1.RecordSelectorWidth +
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).Width +
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(1).Width
        .Top = Me.C1TrueDBGrid1.Top +
Me.C1TrueDBGrid1.RowTop(Me.C1TrueDBGrid1.Row)
        .Visible = True
        .Select()
    End With
End Sub

```

To write code in C#

C#

```

private void c1TrueDBGrid1_ButtonClick(object sender,
C1.Win.C1TrueDBGrid.ColEventArgs e)
{
    this.listBox1.Left = this.c1TrueDBGrid1.Left +
this.c1TrueDBGrid1.RecordSelectorWidth +
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].Width +
this.c1TrueDBGrid1.Splits[0].DisplayColumns[1].Width;
    this.listBox1.Top = this.c1TrueDBGrid1.Top +
this.c1TrueDBGrid1.RowTop(this.c1TrueDBGrid1.Row);
    this.listBox1.Visible = true;
    this.listBox1.Select();
}

```

4. In the **DoubleClick** event of ListBox1, add the following code. When an item is selected in ListBox1, this code will copy its index to the proper column in C1TrueDBGrid1, then make ListBox1 invisible.

To write code in Visual Basic

Visual Basic

```

Private Sub ListBox1_DoubleClick(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ListBox1.DoubleClick
    Me.C1TrueDBGrid1.Columns("CustType").Text = Me.ListBox1.SelectedIndex + 1
    Me.ListBox1.Visible = False
End Sub

```

To write code in C#

C#

```

private void listBox1_DoubleClick(object sender, System.EventArgs e)
{
    this.c1TrueDBGrid1.Columns["CustType"].Text = (this.listBox1.SelectedIndex +
1).ToString();
    this.listBox1.Visible = false;
}

```

5. Then in the **Leave** event of ListBox1, add the following code to make sure the listbox becomes invisible once the selection has been made:

To write code in Visual Basic

Visual Basic

```
Private Sub ListBox1_Leave(ByVal sender As Object, ByVal e As System.EventArgs)
Handles ListBox1.Leave
    Me.ListBox1.Visible = False
End Sub
```

To write code in C#

C#

```
private void listBox1_Leave(object sender, System.EventArgs e)
{
    this.listBox1.Visible = false;
}
```

6. Then in the [Scroll](#) event of C1TrueDBGrid1, add the following code to make sure the listbox becomes invisible once the grid is scrolled:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_Scroll(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.CancelEventArgs) Handles C1TrueDBGrid1.Scroll
    Me.ListBox1.Visible = False
End Sub
```

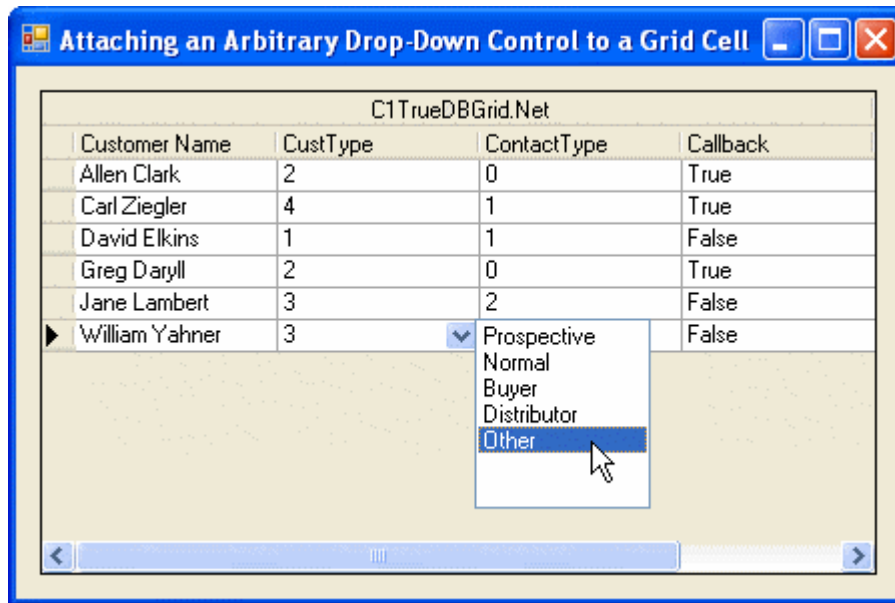
To write code in C#

C#

```
private void c1TrueDBGrid1_Scroll(object sender,
C1.Win.C1TrueDBGrid.CancelEventArgs e)
{
    this.listBox1.Visible = false;
}
```

Run the program and observe the following:

- C1TrueDBGrid1 displays data from the joined table as in [Tutorial 6: Defining Unbound Columns in a Bound Grid](#).
- Click a cell in the *CustType* column to make it the current cell as indicated by the highlight. A button will be displayed at the right edge of the cell. Click the button to fire the [ButtonClick](#) event. ListBox1 will drop down at the right edge of the cell as shown in the following illustration.



- Use the mouse or the UP ARROW and DOWN ARROW keys to move the highlight bar of ListBox1. If another cell in the grid is clicked, ListBox1 will lose focus and become invisible.
- Double-click any item in ListBox1. The current cell in the grid will be updated with the selected item, and ListBox1 will disappear until editing is again initiated.
- If the current cell is moved to another column, the button will disappear from the cell in the *CustType* column.

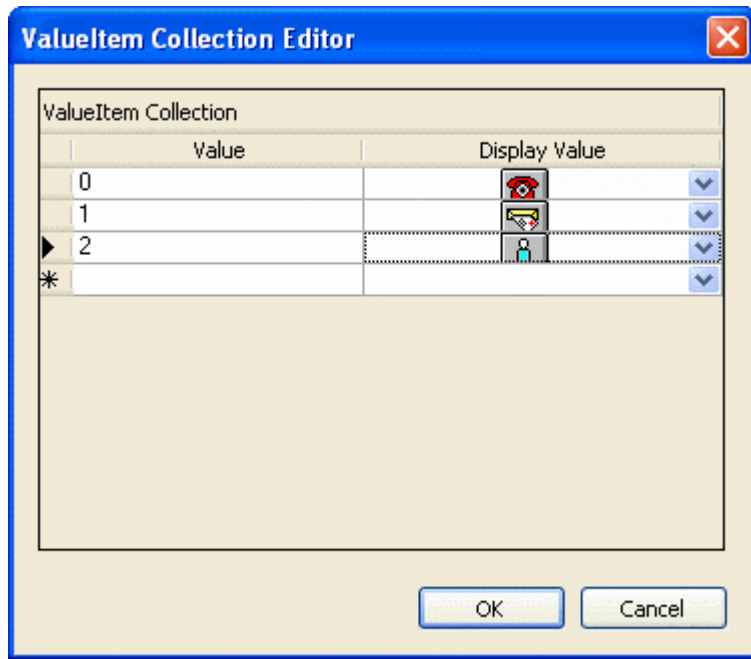
You've successfully completed attaching an arbitrary drop-down control to a grid cell; this concludes tutorial 9.

Tutorial 10: Enhancing the User Interface with In-Cell Bitmaps

In this tutorial, you will learn how to use the [ValueItems](#) object and its collection of [ValueItem](#) objects to display bitmaps and check boxes in a cell—without writing a single line of code!

Complete the following steps:

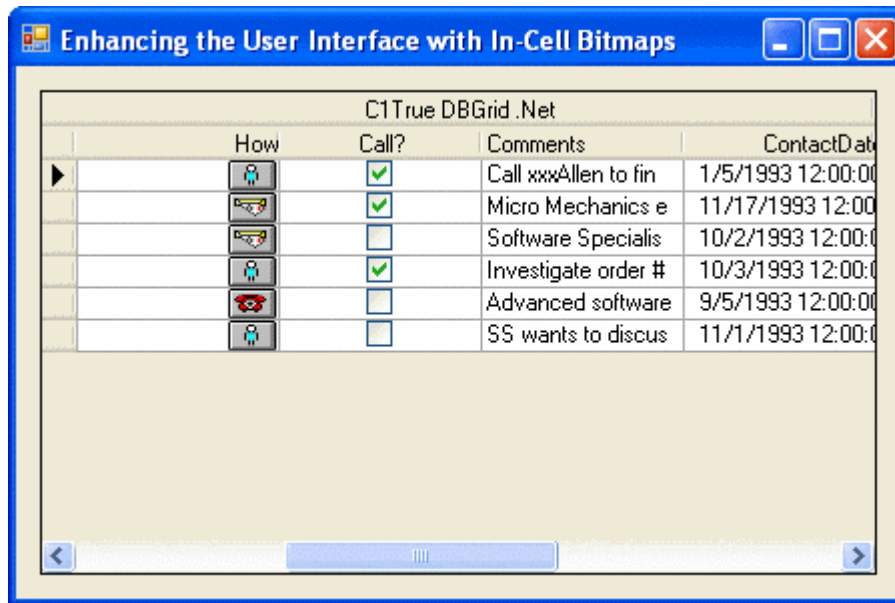
1. Start with the project used in [Tutorial 7: Displaying Translated Data with the Built-In Combo](#).
2. First, change the captions of the *ContactType* and *Callback* columns. Open up the **C1TrueDBGrid Designer** by clicking on the **ellipsis** button next to the **Columns** property in the Properties window.
3. Select the *ContactType* column, then in the left pane, change its **Caption** property to "How". Then in a similar manner, change the *CallBack* column caption to "Call?"
4. Change the [HorizontalAlignment](#) property of these two columns by clicking the **Align center** button so that the bitmaps will be centered within each cell. Open the **SplitCollection Editor** by clicking on the **ellipsis** button next to the **Splits** property in the Properties window. Next open the **C1DisplayColumnCollection Editor** by clicking on the **ellipsis** next to the **DisplayColumn** property in the Splits editor. Select the *How* column in the left pane, then in the right pane, click the expand icon next to the **Styles** property. Under the **Styles** object for this column set the [HorizontalAlignment](#) property to **Center**. Then set the [VerticalAlignment](#) properties to **Center**. In a similar manner, set the [HorizontalAlignment](#) and [VerticalAlignment](#) properties for the *Call?* column.
5. Next, assign bitmaps and check boxes to selected columns by populating the corresponding [ValueItems](#) object. We will start with the bitmaps in column 2. Open up the **C1TrueDBGrid Designer** by clicking on the **ellipsis** button next to the **Columns** property in the Properties window. Select the *How* column, and then in the left pane, click the expand icon next to the **ValueItem** object. Open up the **ValueItemCollection Editor** by clicking on the **ellipsis** button next to the **Values** property:



6. Create three new ValueItem objects by clicking the **Add** button in the left pane. The possible values of the *ContactType* field are 0, 1, and 2, which represent telephone, mail, and personal contact, respectively. Bitmaps shall be displayed in the cell instead of these numeric values. If the full product is installed, the following files will be found in the Tutor10 subdirectory of the Tutorials installation directory: PHONE.BMP, MAIL.BMP, and PERSON.BMP.
7. In the right pane, for the first **ValueItem**, enter 0 as the value, then in the **DisplayValue** property box, click the **ellipsis** button to search for the Image file to display in the cell. Locate the Phone.bmp file in the Tutor10 subdirectory of the **WinForms Edition** installation directory. In a similar manner set the other two **ValueItem** objects to a Value of 1, DisplayValue of Mail.bmp, and a Value of 2, DisplayValue of Person.BMP, respectively. Return to the **ValueItems** object in the **C1TrueDBGrid Designer** and set the **Translate** and **CycleOnClick** properties equal to **True**.
8. To set the check boxes for column 3, in the **C1TrueDBGrid Designer**, select the *Call* column. In the left pane, expand the **ValueItems** object and set the **Presentation** property to **CheckBox**. This will display the column's Boolean values as check boxes. Then finally, under the same object set the **Translate** and **CycleOnClick** properties to **True**.

Run the program and observe the following:

- C1TrueDBGrid1 displays data from the joined table.
- The *How* and *Call?* columns now display bitmaps instead of numeric values as shown in the following figure.



- Click a cell in the *How* column to make it the current cell. Then click it again several times and observe how the cell cycles through the PHONE, MAIL, and PERSON bitmaps.
- Click a cell in the *Call?* column and observe how the cell cycles through the check box bitmaps.

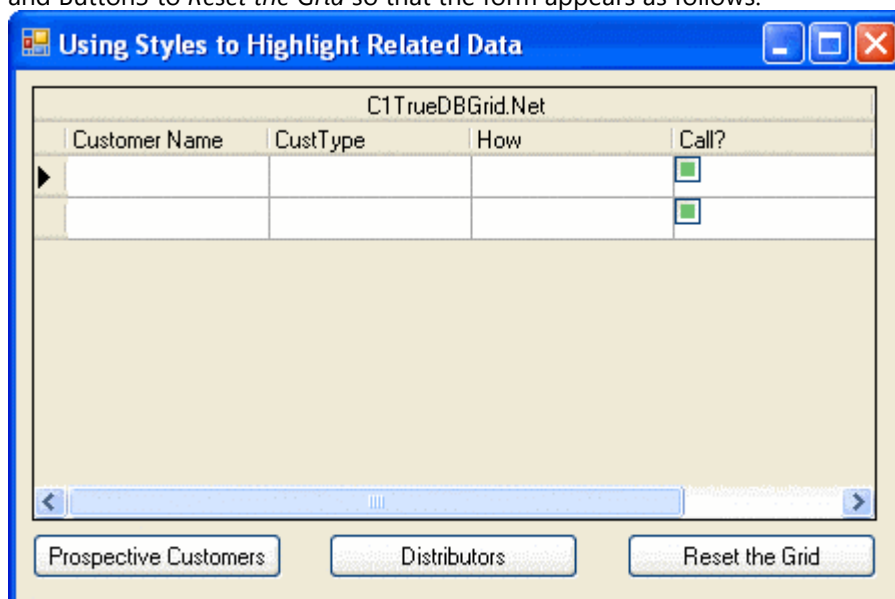
You've successfully completed enhancing the user interface with in-cell bitmaps; this concludes tutorial 10.

Tutorial 11: Using Styles to Highlight Related Data

In this tutorial, you will learn how to change the grid's display to highlight rows by creating row styles depending upon a value in the grid. **True DBGrid** uses the **FetchRowStyle** event to create style characteristics and apply them to rows dynamically.

Complete the following steps:

1. Start with the project used in [Tutorial 10: Enhancing the User Interface with In-Cell Bitmaps](#).
2. Add three buttons to the form. Change the caption of Button1 to *Prospective Customers*, Button2 to *Distributors*, and Button3 to *Reset the Grid* so that the form appears as follows.



3. Add the following declarations to the General section of Form1:

To write code in Visual Basic

```
Visual Basic  
Dim bflag As Integer
```

To write code in C#

```
C#  
int bflag;
```

4. Enter the following code in the **Click** event of Button1:

To write code in Visual Basic

```
Visual Basic  
  
' Prospective Customers.  
Me.C1TrueDBGrid1.FetchRowStyles = True  
bFlag = 1  
Me.C1TrueDBGrid1.Refresh()
```

To write code in C#

```
C#  
  
// Prospective Customers.  
this.c1TrueDBGrid1.FetchRowStyles = true;  
bFlag = 1;  
this.c1TrueDBGrid1.Refresh();
```

5. Enter the following code in the **Click** event of Button2:

To write code in Visual Basic

```
Visual Basic  
  
' Distributors.  
Me.C1TrueDBGrid1.FetchRowStyles = True  
bFlag = 2  
Me.C1TrueDBGrid1.Refresh()
```

To write code in C#

```
C#  
  
// Distributors.  
this.c1TrueDBGrid1.FetchRowStyles = true;  
bFlag = 2;  
this.c1TrueDBGrid1.Refresh();
```

6. Enter the following code in the **Click** event of Button3:

To write code in Visual Basic

```
Visual Basic
```

```
' Reset the grid.
Me.C1TrueDBGrid1.FetchRowStyles = False
Me.C1TrueDBGrid1.Refresh()
```

To write code in C#

```
C#

// Reset the grid.
this.c1TrueDBGrid1.FetchRowStyles = false;
this.c1TrueDBGrid1.Refresh();
```

7. Next enter the following code into the **FetchRowStyles** event. This code interacts with the setting of the **FetchRowStyles** property in the click event. When the **FetchRowStyles** is set to **True**, the grid fires the **FetchRowStyle** event when it needs to repaint the cells. Thus the row style is applied according to the value of the bflag flag integer:

To write code in Visual Basic

```
Visual Basic

Private Sub C1TrueDBGrid1_FetchRowStyle(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.FetchRowStyleEventArgs) Handles C1TrueDBGrid1.FetchRowStyle

    If bFlag = 1 And Me.C1TrueDBGrid1 (e.Row, "CustType") = 1 Then
        Dim fntFont As New Font(e.CellStyle.Font.Name, e.CellStyle.Font.Size,
FontStyle.Bold)
        e.CellStyle.Font = fntFont
        e.CellStyle.ForeColor = System.Drawing.Color.Blue
    End If

    If bFlag = 2 And Me.C1TrueDBGrid1 (e.Row, "CustType") = 4 Then
        e.CellStyle.ForeColor = System.Drawing.Color.White
        e.CellStyle.BackColor = System.Drawing.Color.Red
    End If
End Sub
```

To write code in C#

```
C#

private void C1TrueDBGrid1_FetchRowStyle(object sender,
C1.Win.C1TrueDBGrid.FetchRowStyleEventArgs e)
{
    if (bFlag == 1 && (int)this.c1TrueDBGrid1 [e.Row, "CustType"] == 1 )
    {
        Font fntFont = new Font(e.CellStyle.Font.Name, e.CellStyle.Font.Size,
FontStyle.Bold);
        e.CellStyle.Font = fntFont;
        e.CellStyle.ForeColor = System.Drawing.Color.Blue;
    }

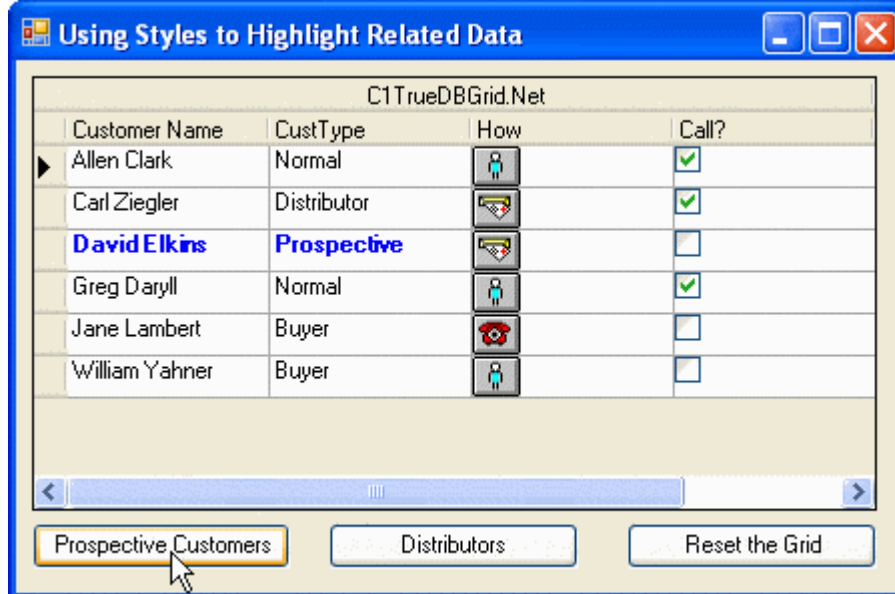
    if (bFlag == 2 && this.c1TrueDBGrid1 [e.Row, "CustType"] == 4 )
    {
```



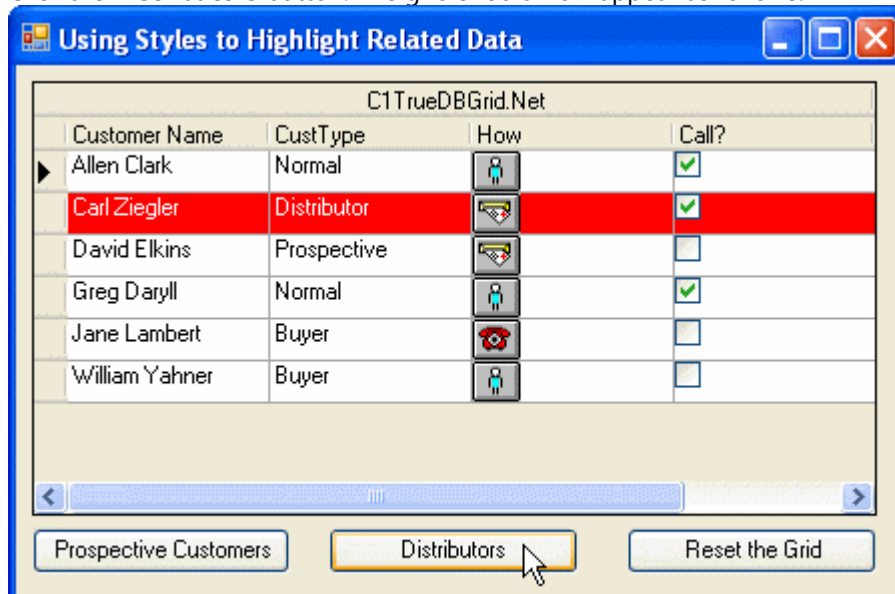
```
e.CellStyle.ForeColor = System.Drawing.Color.White;  
e.CellStyle.BackColor = System.Drawing.Color.Red;  
}  
}
```

Run the program and observe the following:

- C1TrueDBGrid1 displays data as in [Tutorial 10: Enhancing the User Interface with In-Cell Bitmaps](#).
- Click the **Prospective Customers** button. The grid should appear as follows.



- Click the **Distributors** button. The grid should now appear as follows:



- Finally, click the **Reset the Grid** button. The grid should now clear itself of the styles.

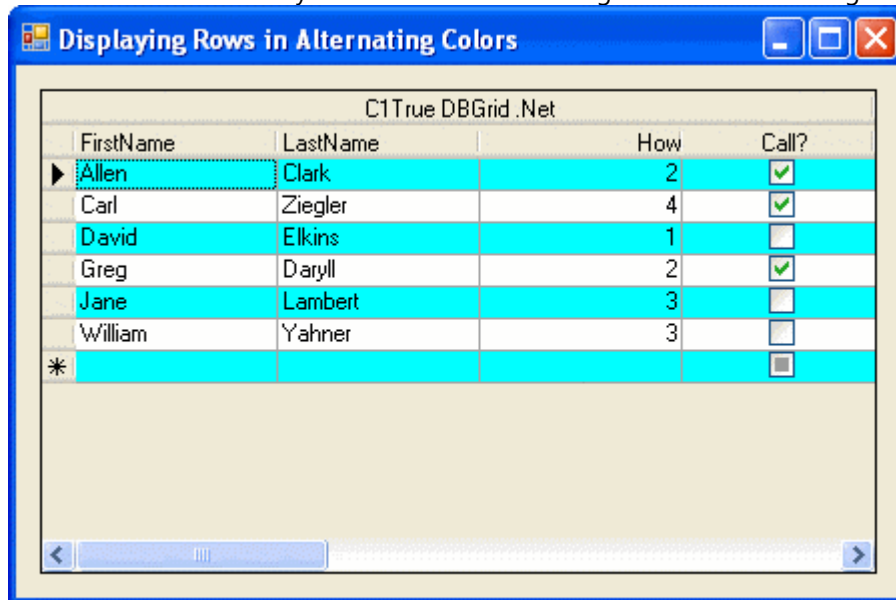
You've successfully completed using styles to highlight related data; this concludes tutorial 11.

Tutorial 12: Displaying Rows in Alternating Colors

In this tutorial, you will learn how to use the [AlternatingRows](#) property and built-in styles to apply alternating colors to grid rows to improve their readability.

Complete the following steps:

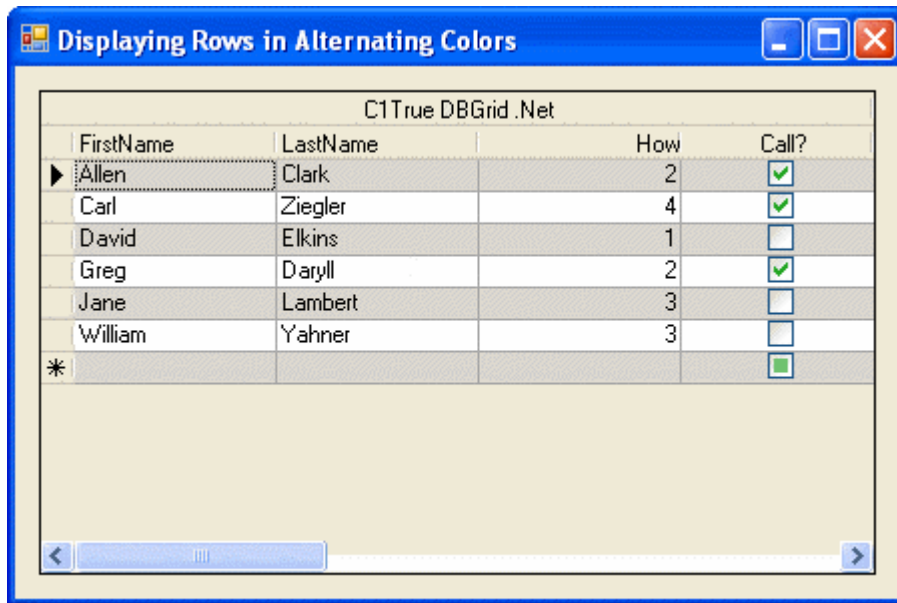
1. Start with the project used in [Tutorial 10: Enhancing the User Interface with In-Cell Bitmaps](#).
2. In the Properties window, set the **AlternatingRows** property to **True**. The grid has default settings for both the **EvenRow** and **OddRow** styles. Use the default settings first and then change the settings for the **EvenRowStyle**.



3. Run the program and observe that C1TrueDBGrid1 displays data as in [Tutorial 10: Enhancing the User Interface with In-Cell Bitmaps](#) except that even-numbered rows have a light cyan background.
4. Click the **ellipsis** button next to the **Styles** property in the Properties window. This will bring up the **C1TrueDBGrid Style Editor**.
5. Select the **EvenRowStyle** in the left pane, and in the right pane change its **BackColor** to **LightGray**. Click **OK** and close the editor.

Run the program and observe the following:

C1TrueDBGrid1 displays data as in the previous image, except that even-numbered rows now have a light gray background:



This concludes the tutorial.

Tutorial 13: Implementing Drag-and-Drop Functionality

In this tutorial, you will learn how to implement drag-and-drop functionality in **True DBGrid for WinForms**.

Set up the True DBGrid for WinForms controls

Complete the following steps:

1. Start a new .NET project
2. Place two [C1TrueDBGrid](#) controls (C1TrueDBGrid1, C1TrueDBGrid2) onto the form. Also add three labels onto the form and arrange them to look like the picture below.
3. In the **C1TrueDBGrid Tasks** menu for C1TrueDBGrid1, locate the **Choose Data Source** drop-down and select **Add Project Data Source**. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select all fields in the **Customers** table and type "DsCustomers" into the **DataSet name** box, and then finish out the wizard.
4. In the **C1TrueDBGrid Tasks** menu for C1TrueDBGrid2, locate the **Choose Data Source** drop-down and select **Add Project Data Source**. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select all fields in the **CallList** table and type "DsCallList" into the **DataSet name** box, and then finish out the wizard.
5. In the general section of the form, add the following declarations:

To write code in Visual Basic

Visual Basic

```
Dim _ptStartDrag As Point
Dim _dragRow As Long
```

To write code in C#

C#

```
Point _ptStartDrag;
long _dragRow;
```

6. Visual Studio adds the following code to the **Load** event of Form1 to fill the new datasets:

To write code in Visual Basic

Visual Basic

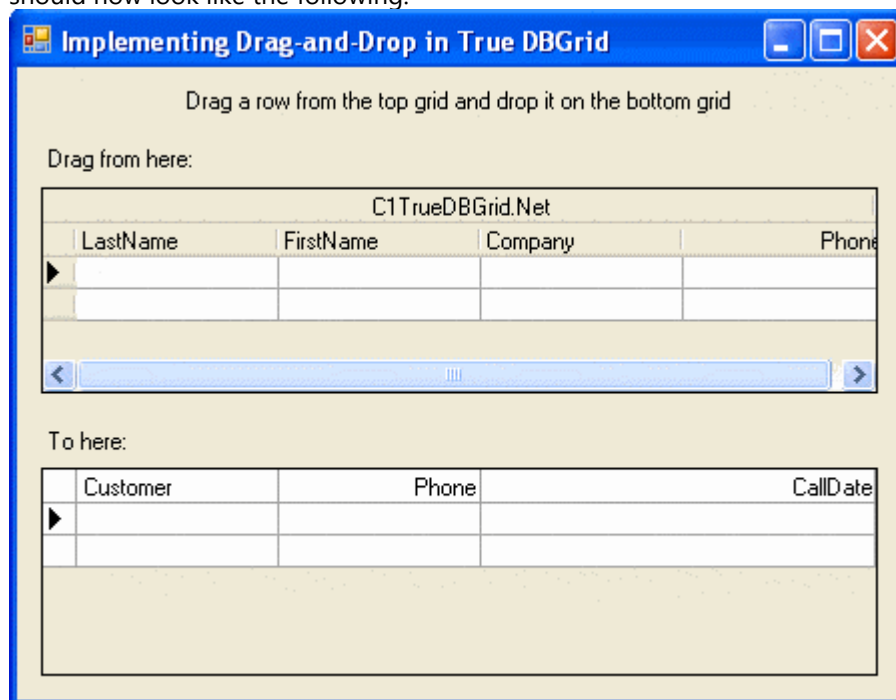
```
Me.CallListTableAdapter.Fill(Me.DsCallList.CallList)
Me.CustomersTableAdapter.Fill(Me.DsCustomers.Customers)
```

To write code in C#

C#

```
this.CallListTableAdapter.Fill(this.DsCallList.CallList);
this.CustomersTableAdapter.Fill(this.DsCustomers.Customers);
```

7. For the first grid (C1TrueDBGrid1) set the **AllowDrag** property to **True**. While, for the second grid, set the **AllowDrop** property to **True**.
8. Right-click C1TrueDBGrid1 and select **Retrieve Fields**. Do the same with the other grid.
9. Open up the **C1TrueDBGrid Designer** for C1TrueDBGrid1 by clicking on the **ellipsis** button next to the **Columns** property for the **C1TrueDBGrid** in the Properties window.
10. Remove all of the columns from the grid except *LastName*, *FirstName*, *Company*, and *Phone* by clicking the **Remove Column** button for each column to remove. Enter the **C1TrueDBGrid Designer** for the other grid and remove all of its columns except *Customer*, *Phone*, and *CallDate*.
11. In the Properties window set the **MarqueeStyle** for C1TrueDBGrid1 to **SolidCellBorder**. In the **C1TrueDBGrid Designer** set Column 3's (Phone) **NumberFormat** property to "(###)###-####". Open up the **C1TrueDBGrid Designer** for the second grid and set its Column 2's **NumberFormat** property to "(###)###-####". The grid should now look like the following:



Add code to your project

This section describes the code needed to drag the contents of a cell or row from C1TrueDBGrid1 to C1TrueDBGrid2. The code assumes that if you want to drag the entire row of data to C1TrueDBGrid2 in order to add a new record there.

1. Add the following subroutine to the project to reset the [MarqueeStyle](#) property of each grid, which is used to provide visual feedback while dragging is in progress. The reset routine will be called to perform clean-up after a drag-and-drop operation concludes.

To write code in Visual Basic

Visual Basic

```
Private Sub ResetDragDrop()  
    ' Turn off drag-and-drop by resetting the highlight and label text.  
    Me._ptStartDrag = Point.Empty  
    Me._dragRow = - 1  
    Me.C1TrueDBGrid1.MarqueeStyle =  
C1.Win.C1TrueDBGrid.MarqueeEnum.SolidCellBorder  
    Me.C1TrueDBGrid2.MarqueeStyle =  
C1.Win.C1TrueDBGrid.MarqueeEnum.SolidCellBorder  
    Me.Label3.Text = "Drag a row from the top grid and drop it on the bottom  
grid."  
End Sub
```

To write code in C#

C#

```
private void ResetDragDrop()  
{  
    // Turn off drag-and-drop by resetting the highlight and label text.  
    this._ptStartDrag = Point.Empty;  
    this._dragRow = - 1;  
    this.c1TrueDBGrid1.MarqueeStyle =  
C1.Win.C1TrueDBGrid.MarqueeEnum.SolidCellBorder;  
    this.c1TrueDBGrid2.MarqueeStyle =  
C1.Win.C1TrueDBGrid.MarqueeEnum.SolidCellBorder;  
    this.label3.Text = "Drag a row from the top grid and drop it on the bottom  
grid.";  
}
```

2. Enter the following code to handle the mouse related events:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_MouseDown(ByVal sender As Object, ByVal e As  
System.Windows.Forms.MouseEventArgs) Handles C1TrueDBGrid1.MouseDown  
    Dim row, col As Integer  
    Me._ptStartDrag = Point.Empty  
    Me._dragRow = - 1  
    If Me.C1TrueDBGrid1.CellContaining(e.X, e.Y, row, col) Then  
  
        ' Save the starting point of the drag operation.
```

```

        Me._ptStartDrag = New Point(e.X, e.Y)
        Me._dragRow = row
    End If
End Sub

Private Sub C1TrueDBGrid1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles C1TrueDBGrid1.MouseMove

    ' If we don't have an empty start drag point, then the drag has been
    initiated.
    If Not Me._ptStartDrag.IsEmpty Then

        ' Create a rectangle that bounds the start of the drag operation by 2
        pixels.
        Dim r As New Rectangle(Me._ptStartDrag, Size.Empty)
        r.Inflate(2, 2)

        ' If we've moved more than 2 pixels, start the drag operation.
        If Not r.Contains(e.X, e.Y) Then
            Me.C1TrueDBGrid1.Row = Me._dragRow
            Me.C1TrueDBGrid1.MarqueeStyle =
C1.Win.C1TrueDBGrid.MarqueeEnum.HighlightRow
            Me.C1TrueDBGrid1.DoDragDrop(Me._dragRow, DragDropEffects.Copy)
        End If
    End If
End Sub

```

To write code in C#

```

C#

private void C1TrueDBGrid1_MouseDown(object sender,
System.Windows.Forms.MouseEventArgs e)
{
    int row, col;
    this._ptStartDrag = Point.Empty;
    this._dragRow = - 1;
    if (this.C1TrueDBGrid1.CellContaining(e.X, e.Y, row, col) )
    {
        // Save the starting point of the drag operation.
        this._ptStartDrag = new Point(e.X, e.Y);
        this._dragRow = row;
    }
}

private void C1TrueDBGrid1_MouseMove(object sender,
System.Windows.Forms.MouseEventArgs e)
{
    // If we don't have an empty start drag point, then the drag has been
    initiated.
    if (!this._ptStartDrag.IsEmpty )
    {

```

```

        // Create a rectangle that bounds the start of the drag operation by 2
pixels.
Rectangle r = new Rectangle(this._ptStartDrag, Size.Empty);
r.Inflate(2, 2);

// If we've moved more than 2 pixels, start the drag operation.
if (!r.Contains(e.X, e.Y) )
{
    this.clTrueDBGrid1.Row = this._dragRow;
    this.clTrueDBGrid1.MarqueeStyle =
C1.Win.C1TrueDBGrid.MarqueeEnum.HighlightRow;
    this.clTrueDBGrid1.DoDragDrop(this._dragRow, DragDropEffects.Copy);
}
}
}

```

3. Enter the following code to handle the dragging and dropping events:

To write code in Visual Basic

Visual Basic

```

Private Sub C1TrueDBGrid2_DragEnter(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles C1TrueDBGrid2.DragEnter
    Me.Label3.Text = "Create a new record when dropped..."
    e.Effect = DragDropEffects.Copy
End Sub

Private Sub C1TrueDBGrid2_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles C1TrueDBGrid2.DragDrop
    Try
        Dim row As Integer = CInt(e.Data.GetData(GetType(Integer)))

        ' Use the grid's indexer to get some data.
        Dim custname As String = Me.C1TrueDBGrid1(row, "FirstName").ToString()

        ' Use the CellText() method to get some data.
        custname += " " + Me.C1TrueDBGrid1.Columns("LastName").CellText(row)

        ' Use the CellValue() method to get some data.
        custname += " " +
Me.C1TrueDBGrid1.Columns("Company").CellValue(row).ToString()

        ' Add a new row to the data set for the bottom grid.
        Dim drv As DataRowView = Me.DsCallList.CallList.DefaultView.AddNew()
        drv("CallDate") = DateTime.Now
        drv("Customer") = custname
        drv("Phone") = Me.C1TrueDBGrid1.Columns("Phone").Value.ToString()
        drv.EndEdit()
        Me.C1TrueDBGrid2.MoveLast()
        Me.C1TrueDBGrid2.Select()

        ' Commit changes to the database.

```

```

        Dim inserted As DataSet = Me.DsCallList.GetChanges(DataRowState.Added)
        If Not (inserted Is Nothing) Then
            Me.CallListTableAdapter.Update(inserted)
        End If
        Me.ResetDragDrop()
    Catch ex As System.Exception
        MessageBox.Show(ex.Message)
    End Try
End Sub

```

To write code in C#

```

C#

private void C1TrueDBGrid2_DragEnter(object sender,
System.Windows.Forms.DragEventArgs e)
{
    this.label3.Text = "Create a new record when dropped...";
    e.Effect = DragDropEffects.Copy;
}

private void C1TrueDBGrid2_DragDrop(object sender,
System.Windows.Forms.DragEventArgs e)
{
    try
    {
        int row = (int)e.Data.GetData(typeof(int));

        // Use the grid's indexer to get some data.
        string custname = this.c1TrueDBGrid1[row, "FirstName"].ToString();

        // Use the CellText() method to get some data.
        custname += " " + this.c1TrueDBGrid1.Columns["LastName"].CellText(row);

        // Use the CellValue() method to get some data.
        custname += " " +
this.c1TrueDBGrid1.Columns["Company"].CellValue(row).ToString();

        // Add a new row to the data set for the bottom grid.
        DataRowView drv = this.DsCallList.CallList.DefaultView.AddNew();
        drv["CallDate"] = DateTime.Now;
        drv["Customer"] = custname;
        drv["Phone"] = this.c1TrueDBGrid1.Columns["Phone"].Value.ToString();
        drv.EndEdit();
        this.c1TrueDBGrid2.MoveLast();
        this.c1TrueDBGrid2.Select();

        // Commit changes to the database.
        DataSet inserted = this.DsCallList.GetChanges(DataRowState.Added);
        if (! (inserted == null) )
        {
            this.CallListTableAdapter.Update(inserted);
        }
    }
    catch { }
}

```



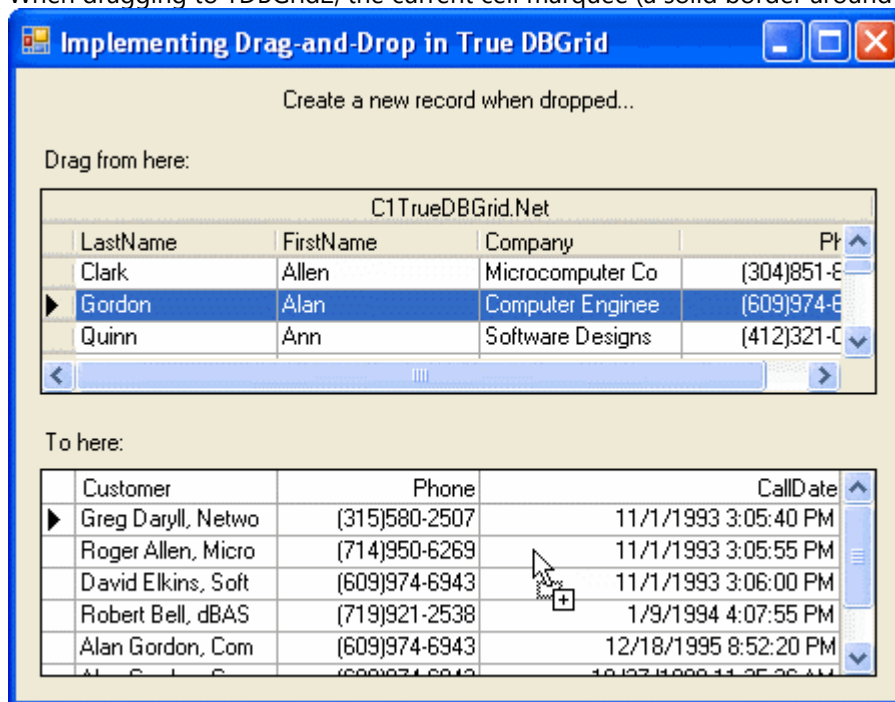
```

    }
    this.ResetDragDrop();
}
catch (System.Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

```

Run the program and observe the following:

- If an item in a column in C1TrueDBGrid1 is dragged, the entire row in C1TrueDBGrid1 is highlighted, indicating that the entire row of data is being dragged.
- When dragging to TDBGrid2, the current cell marquee (a solid border around the cell) disappears.



- If the data is dropped on C1TrueDBGrid2, a new record is created using the data from the current row of C1TrueDBGrid1.

You've successfully completed implementing drag-and-drop in **C1TrueDBGrid**; this concludes tutorial 13.

Tutorial 14: Creating a Grid with Fixed, Nonscrolling Columns

Often, you would like to prevent one or more columns from scrolling horizontally or vertically so that they will always be in view. The [SplitCollection](#) of **True DBGrid** provides a generalized mechanism for defining groups of adjacent columns, and can be used to implement any number of fixed, nonscrolling columns or rows. In this tutorial, you will learn how to write code to create a grid with two horizontal splits, and then "fix" a pair of columns in the leftmost split.

Complete the following steps:

1. Follow [Tutorial 1: Binding True DBGrid to a DataSet](#) to create a project with a C1TrueDBGrid bound to a DataSet.
2. In the **Load** event for Form1, place the following code to create an additional split and to fix columns 0 and 1 in the leftmost split:

To write code in Visual Basic**Visual Basic**

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    ' Create an additional split.
    Me.C1TrueDBGrid1.InsertHorizontalSplit(0)

    ' Hide all columns in the leftmost split except 0 and 1.
    Dim x As Integer
    For x = 2 To Me.C1TrueDBGrid1.Columns.Count - 1
        Me.C1TrueDBGrid1.Splits(0).DisplayColumns(x).Visible = False
    Next

    ' Configure split 0 to display exactly 2 columns.
    With Me.C1TrueDBGrid1.Splits(0)
        .SplitSizeMode = C1.Win.C1TrueDBGrid.SizeModeEnum.NumberOfColumns
        .SplitSize = 2
        .AllowHorizontalSizing = False
    End With

    ' Make columns 0 and 1 invisible in split 1.
    Me.C1TrueDBGrid1.Splits(1).DisplayColumns(0).Visible = False
    Me.C1TrueDBGrid1.Splits(1).DisplayColumns(1).Visible = False

    ' Turn off record selectors in split 1.
    Me.C1TrueDBGrid1.Splits(1).RecordSelectors = False
End Sub
```

To write code in C#**C#**

```
private void Form1_Load(System.Object sender, System.EventArgs e)
{
    // Create an additional split.
    this.c1TrueDBGrid1.InsertHorizontalSplit(0);

    // Hide all columns in the leftmost split except 0 and 1.
    int x;
    for (x = 2 ; x < this.c1TrueDBGrid1.Columns.Count; x++)
    {
        this.c1TrueDBGrid1.Splits[0].DisplayColumns[x].Visible = false;
    }

    // Configure split 0 to display exactly 2 columns.
    this.c1TrueDBGrid1.Splits[0].SplitSizeMode =
```

```

C1.Win.C1TrueDBGrid.SizeModeEnum.NumberOfColumns;
    this.c1TrueDBGrid1.Splits[0].SplitSize = 2;
    this.c1TrueDBGrid1.Splits[0].AllowHorizontalSizing = false;

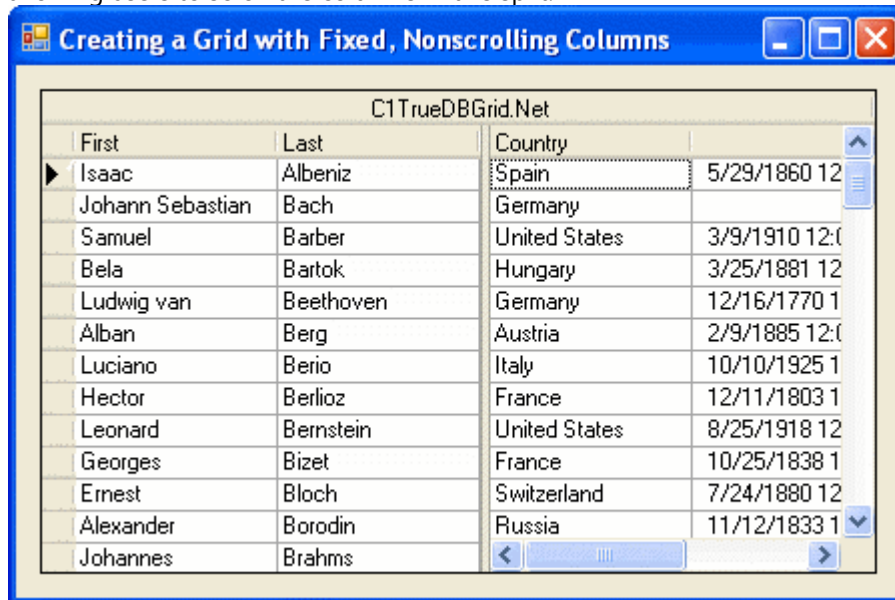
    // Make columns 0 and 1 invisible in split 1.
    this.c1TrueDBGrid1.Splits[1].DisplayColumns[0].Visible = false;
    this.c1TrueDBGrid1.Splits[1].DisplayColumns[1].Visible = false;

    // Turn off record selectors in split 1.
    this.c1TrueDBGrid1.Splits[1].RecordSelectors = false;
}

```

Run the program and observe the following:

- **C1TrueDBGrid** displays data from the Data control as in [Tutorial 1: Binding True DBGrid to a DataSet](#).
- The two columns (*First* and *Last*) in the leftmost split are fixed and cannot be scrolled. In fact, there is no horizontal scroll bar present under the left split. A horizontal scroll bar appears under the rightmost split, allowing users to scroll the columns in this split.



Use splits to create fixed, non-scrolling columns anywhere within the grid – even in the middle. Also use splits to present different views of data. For example, splits can be created that scroll independently (in the vertical direction) so that users may compare records at the beginning of the database with those at the end. For more information, see [How to Use Splits](#).

You've successfully completed creating a grid with fixed, nonscrolling columns; this concludes tutorial 14.

Tutorial 15: Using PrintInfo and Print Preview

In this tutorial, you will learn how to use the printing and exporting capabilities of **True DBGrid for WinForms**.

Complete the following steps:

1. Start with the project created in [Tutorial 1: Binding True DBGrid to a DataSet](#).
2. Add one Button to the form (Button1) and change its **Text** property to "Print Preview".
3. Enter the following code in the **Load** event of Form1. It changes the **BackColor** of a column, changes a

column's font, sets the **NumberFormat** property for a column to the **FormatText** event, and changes the HeadingStyle:

To write code in Visual Basic

Visual Basic

```
' Change the presentation of the grid.
With Me.ClTrueDBGrid1.Splits(0).DisplayColumns
    .Item("Country").Style.BackColor = System.Drawing.Color.Cyan
    Dim fntFont As Font
    fntFont = New Font("Times New Roman", .Item("Country").Style.Font.Size,
    FontStyle.Regular)
    .Item("Country").Style.Font = fntFont
    .Item("Last").Style.ForeColor = System.Drawing.Color.Red
End With
Me.ClTrueDBGrid1.Columns("last").NumberFormat = "FormatText Event"
With Me.ClTrueDBGrid1.HeadingStyle
    Dim fntfont As Font
    fntfont = New Font(.Font.Name, .Font.Size, FontStyle.Bold)
    .Font = fntfont
    .BackColor = System.Drawing.Color.Blue
    .ForeColor = System.Drawing.Color.Yellow
End With
```

To write code in C#

C#

```
// Change the presentation of the grid.
ClDisplayColumn col = this.clTrueDBGrid1.Splits[0].DisplayColumns["Country"];
col.Style.BackColor = System.Drawing.Color.Cyan;
Font fntFont;
fntFont = new Font("Times new Roman", col.Style.Font.Size, FontStyle.Regular);
col.Style.Font = fntFont;
clTrueDBGrid1.Splits[0].DisplayColumns["Last"].Style.ForeColor =
System.Drawing.Color.Red;
this.clTrueDBGrid1.Columns["last"].NumberFormat = "FormatText event";
Font fntfont;
fntfont = new Font(Font.Name, this.clTrueDBGrid1.HeadingStyle.Font.Size,
FontStyle.Bold);
this.clTrueDBGrid1.HeadingStyle.Font = fntfont;
this.clTrueDBGrid1.HeadingStyle.BackColor = System.Drawing.Color.Blue;
this.clTrueDBGrid1.HeadingStyle.ForeColor = System.Drawing.Color.Yellow;
```

4. In the previous code the **NumberFormat** property for a column was set to **FormatText**. This means that the **FormatText** event will fire enabling the programmer to change the style and format of column values. Enter the following code into the **FormatText** event, which changes the column values to uppercase:

To write code in Visual Basic

Visual Basic

```
Private Sub ClTrueDBGrid1_FormatText(ByVal sender As Object, ByVal e As
Cl.Win.ClTrueDBGrid.FormatTextEventArgs) Handles ClTrueDBGrid1.FormatText
```

```
e.Value = UCase(e.Value)
End Sub
```

To write code in C#**C#**

```
private void ClTrueDBGrid1_FormatText(object sender,
Cl.Win.ClTrueDBGrid.FormatTextEventArgs e)
{
    e.Value = e.Value.ToUpper();
}
```

5. Add the following code to the **Click** event of Button1. It uses the **PrintInfo** object and its properties and methods to create a print page header and footer. It ends by calling the **PrintPreview** method that invokes the Print Preview window:

To write code in Visual Basic**Visual Basic**

```
With Me.ClTrueDBGrid1.PrintInfo
    Dim fntFont As Font
    fntFont = New Font(.PageHeaderStyle.Font.Name, .PageHeaderStyle.Font.Size,
    FontStyle.Italic)
    .PageHeaderStyle.Font = fntFont
    .PageHeader = "Composers Table"

    ' Column headers will be on every page.
    .RepeatColumnHeaders = True

    ' Display page numbers (centered).
    .PageFooter = "Page: \p"

    ' Invoke print preview.
    .UseGridColors = True
    .PrintPreview()
End With
```

To write code in C#**C#**

```
Font fntFont;
fntFont = new Font(this.clTrueDBGrid1.PrintInfo.PageHeaderStyle.Font.Name,
this.clTrueDBGrid1.PrintInfo.PageHeaderStyle.Font.Size, FontStyle.Italic);
this.clTrueDBGrid1.PrintInfo.PageHeaderStyle.Font = fntFont;
this.clTrueDBGrid1.PrintInfo.PageHeader = "Composers Table";

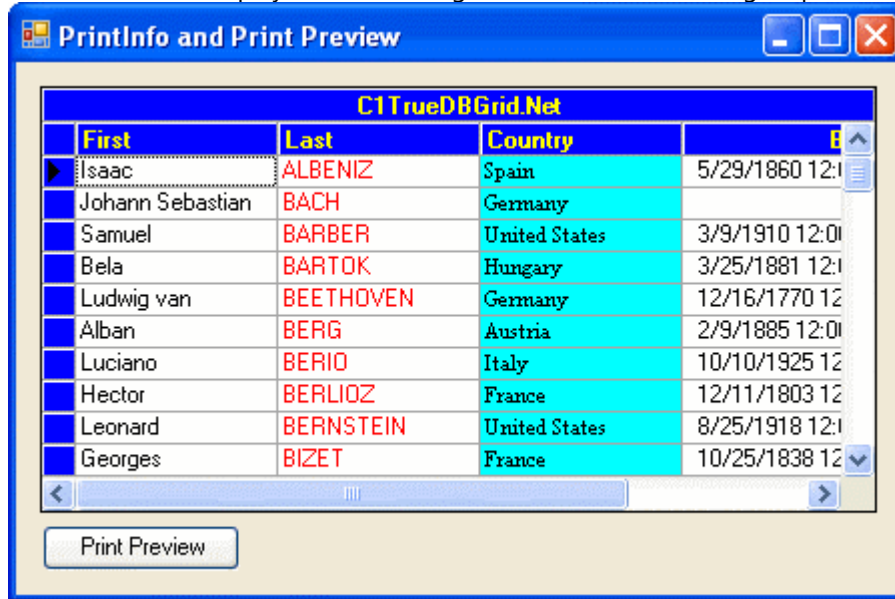
// Column headers will be on every page.
this.clTrueDBGrid1.PrintInfo.RepeatColumnHeaders = true;

// Display page numbers (centered).
this.clTrueDBGrid1.PrintInfo.PageFooter = "Page: \\p";
```

```
// Invoke print preview.
this.clTrueDBGrid1.PrintInfo.UseGridColors = true;
this.clTrueDBGrid1.PrintInfo.PrintPreview();
```

Run the program and observe the following:

- **C1TrueDBGrid1** displays the data using the font and color changes specified in step 4.



- Click the **Print Preview** button to display a separate application window. Note that the output mirrors the format of the grid.

You've successfully completed using PrintInfo and Print Preview; this concludes tutorial 15.

Tutorial 16: Using the Hierarchical Display

In this tutorial, you will learn how to display Master Detail DataSet information through the grid's hierarchical display. This tutorial is similar to [Tutorial 3: Linking Multiple True DBGrid Controls](#), but this tutorial displays the same master detail information as [Tutorial 3: Linking Multiple True DBGrid Controls](#), except it only uses one **C1TrueDBGrid** object.

Complete the following steps:

1. Start by setting up a form with a grid and Master Detail DataSet by following the steps in [Tutorial 3: Linking Multiple True DBGrid Controls](#).
2. In the Properties window, set the **DataSource** property of the grid to **dsMasterDetail** and the **DataMember** property to **Composer**.
3. Next in the **Load** event of Form1 add the following code, which fills both the DataAdapters and sets the grid's display to hierarchical:

To write code in Visual Basic

Visual Basic

```
Me.ComposerTableAdapter.Fill(Me.DsMasterDetail.Composer)
Me.C1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.Hierarchical
```

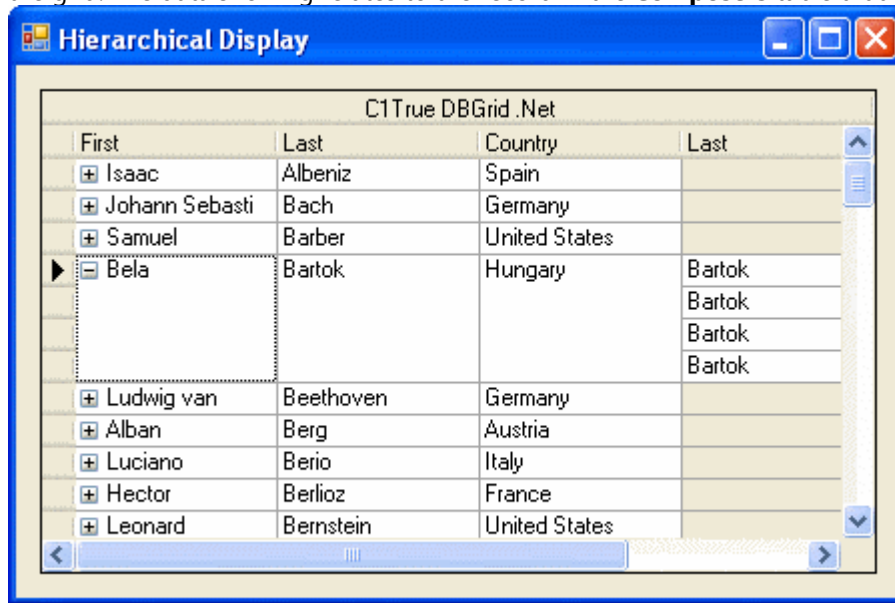
To write code in C#

C#

```
this.ComposerTableAdapter.Fill(this.DsMasterDetail.Composer);
this.c1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.Hierarchical;
```

Run the program and observe the following:

- C1TrueDBGrid1 displays the **Composers** table as it would if it were bound to it, but each row has an expand icon. Expand one of the rows. Notice that the associated *Opus* column data is displayed in the far columns of the grid. The data showing relates to the record in the **Composers** table that was expanded:



You've successfully completed using the hierarchical display; this concludes the tutorial.

Tutorial 17: Creating a Grouping Display

In this tutorial, you will learn how to use the **DataView** property to create a Grouping area above the grid, which enables the user to sort the data by columns at run time.

Complete the following steps:

1. Start with the project created in [Tutorial 1: Binding True DBGrid to a DataSet](#).
2. Add the following code to the **Load** event of Form1 after the current DataAdapter code:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy
```

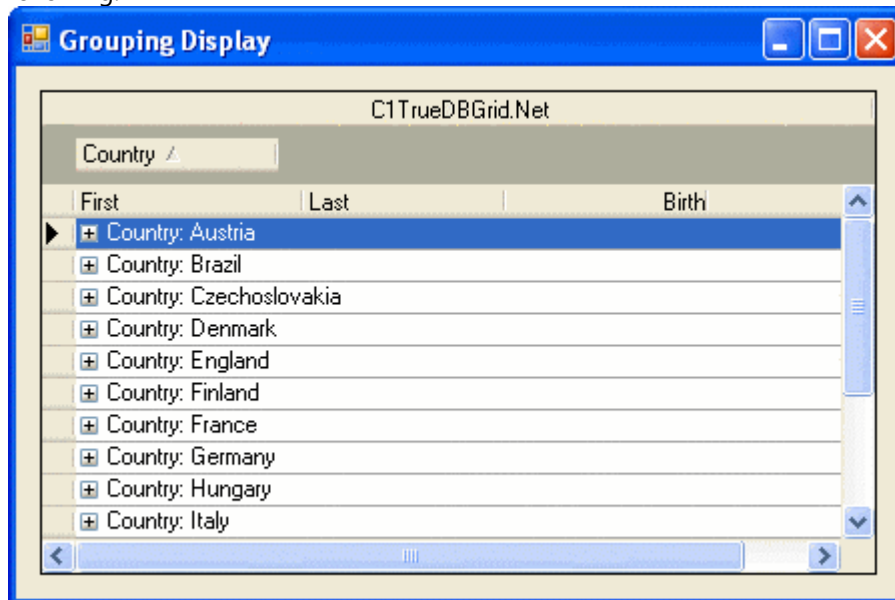
To write code in C#

C#

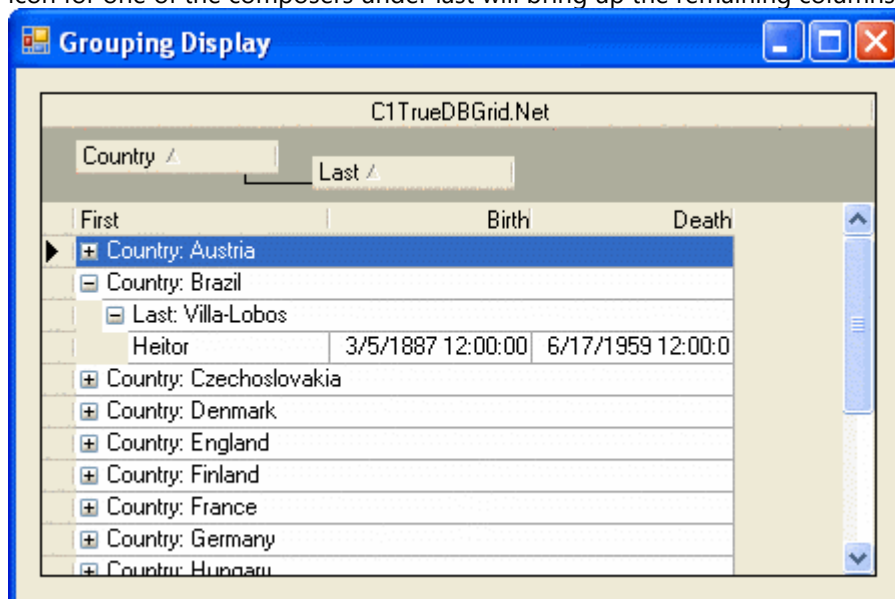
```
this.c1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy;
```

Run the program and observe the following:

- C1TrueDBGrid1 displays the data specified in [Tutorial 1: Binding True DBGrid to a DataSet](#).
- Notice that there is a grouping section above the grid now.
- Click the *Country* column header and drag it into the grouping area. Your grid should now look like the following:



- Notice that the C1TrueDBGrid has placed all of the members of the Country class into the left column and each row has an expand icon. Clicking on the expand icon will show you the data for every composer born in this country.
- Now drag the *Last* column header into the grouping area, then click the expand icon next to Germany. Notice that the data is first sorted by country, then sorted by the last name of the composer. Clicking on the expand icon for one of the composers under last will bring up the remaining columns of data:



You've successfully completed creating a grouping area in the grid; this concludes the tutorial.

Tutorial 18: Using Value Translation

In this tutorial, you will learn how to use the [C1TrueDBDropDowns ValueTranslate](#) property to automatically translate data from the drop-down detail data to the grid's master data.

Complete the following steps:

1. Start with the project created in [Tutorial 8: Attaching a Drop-Down Control to a Grid Cell](#).
2. In the **Load** event of the form add the following code to the existing code:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBDropDown1.ValueTranslate = True
Me.C1TrueDBDropDown1.ListField = "TypeDesc"
Me.C1TrueDBDropDown1.DataField = "TypeID"
```

To write code in C#

C#

```
this.c1TrueDBDropDown1.ValueTranslate = true;
this.c1TrueDBDropDown1.ListField = "TypeDesc";
this.c1TrueDBDropDown1.DataField = "TypeID";
```

Run the program and observe the following:

- C1TrueDBGrid1 displays the data specified in [Tutorial 8: Attaching a Drop-Down Control to a Grid Cell](#).
- The values in the *CustType* column of the grid now display the long descriptions displayed in the drop-down. The values were automatically translated from the drop-down to the grid column at run time.

You've successfully completed using the [C1TrueDBDropDowns ValueTranslate](#) property; this concludes the tutorial.

Tutorial 19: Using Range Selection

In this tutorial, you will learn how to use the **SelectedRows** and **SelectedCols** objects copy a range from the grid in such a format that it can be pasted into Microsoft Excel.

Complete the following steps:

1. Start with the project created in [Tutorial 1: Binding True DBGrid to a DataSet](#).
2. Add a command button to the form, place it in the lower left corner of the form, and set its **Text** property to "Copy".
3. Next add the following code to the **Click** event of Button1:

To write code in Visual Basic

Visual Basic

```
' String to be copied to the clipboard.
Dim strTemp As String

Dim row As Integer
Dim col As C1.Win.C1TrueDBGrid.C1DataColumn
```

```
Dim cols As Integer, rows As Integer
If Me.ClTrueDBGrid1.SelectedRows.Count > 0 Then
    For Each row In Me.ClTrueDBGrid1.SelectedRows

        ' Copy everything here.
        For Each col In Me.ClTrueDBGrid1.SelectedCols
            strTemp = strTemp & col.CellText(row) & vbTab
        Next
        strTemp = strTemp & vbCrLf
    Next
    System.Windows.Forms.Clipboard.SetDataObject(strTemp, False)
    MessageBox.Show ("Range of " & Me.ClTrueDBGrid1.SelectedCols.Count & " x " &
        ClTrueDBGrid1.SelectedRows.Count & " cells have been copied to the clipboard in
        TAB delimited format")
Else
    MessageBox.Show ("Please select a range of cells")
End If
```

To write code in C#**C#**

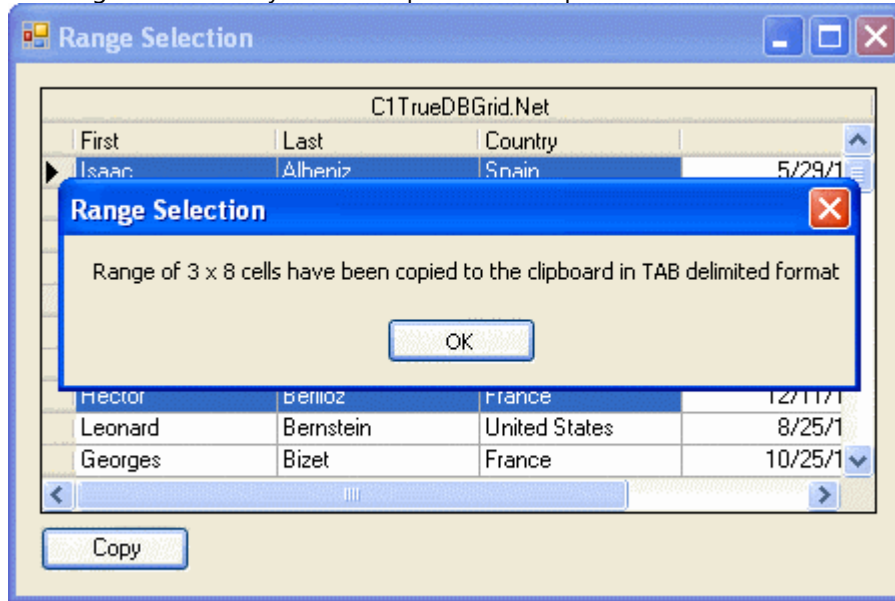
```
// String to be copied to the clipboard.
string strTemp;

int row;
Cl.Win.ClTrueDBGrid.ClDataColumn col;
int cols, rows;
if (this.clTrueDBGrid1.SelectedRows.Count > 0 )
{
    foreach (row in this.clTrueDBGrid1.SelectedRows)
    {

        // Copy everything here.
        foreach (col in this.clTrueDBGrid1.SelectedCols)
        {
            strTemp = strTemp + col.CellText(row) + "\t";
        }
        strTemp = strTemp + "\n";
    }
    System.Windows.Forms.Clipboard.SetDataObject(strTemp, false);
    MessageBox.Show ("Range of " +
this.clTrueDBGrid1.SelectedCols.Count.ToString() + " x " +
this.clTrueDBGrid1.SelectedRows.Count.ToString() + " cells have been copied to
the clipboard in TAB delimited format");
}
else
{
    MessageBox.Show ("Please select a range of cells");
}
```

Run the program and observe the following:

- C1TrueDBGrid1 displays the data specified in [Tutorial 1: Binding True DBGrid to a DataSet](#).
- If you select a range of cells in the **True DBGrid**, then press the copy button a message box will appear detailing the cells that you have copied to the clipboard.



- Now open Microsoft Excel. Select the exact amount of row and column cells as you selected in the **True DBGrid**, then click the **Paste** button. The cells that you copied in the grid are now pasted into Microsoft Excel.

You've successfully completed using range selection; this concludes the tutorial.

Tutorial 20: Displaying Multiple Data Views

In this tutorial, you will learn how to use the grid's **DataView** property to display data in uncommon display formats such as Inverted View, GroupBy View, and Form View.

Complete the following steps:

1. Start with the project created in [Tutorial 1: Binding True DBGrid to a DataSet](#).
2. Add a **ComboBox** control (ComboBox1) to the project, and set its **Text** property to "Data View".
3. In the Properties window, open up the List editor for the ComboBox by clicking on the **ellipsis** button next to the **Items** property. In this editor add the following items:
 Normal
 Inverted
 Form
 GroupBy
 MultipleLines
 Hierarchical
4. Now add the following code to the existing code in the **Load** event of Form1:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.DataView = C1.Win.C1TrueDBGrid.DataViewEnum.Normal
Me.ComboBox1.SelectedIndex = 0
```

To write code in C#**C#**

```
this.clTrueDBGrid1.DataView = Cl.Win.ClTrueDBGrid.DataViewEnum.Normal;
this.comboBox1.SelectedIndex = 0;
```

5. Now add the following code to the **SelectedIndexChanged** event of ComboBox1. It changes the **DataView** property of the grid for each value the user selects in the ComboBox:

To write code in Visual Basic**Visual Basic**

```
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ComboBox1.SelectedIndexChanged
    Select Case ComboBox1.SelectedItem
        Case "Normal"
            Me.ClTrueDBGrid1.DataView = Cl.Win.ClTrueDBGrid.DataViewEnum.Normal
        Case "Inverted"
            Me.ClTrueDBGrid1.DataView = Cl.Win.ClTrueDBGrid.DataViewEnum.Inverted
        Case "Form"
            Me.ClTrueDBGrid1.DataView = Cl.Win.ClTrueDBGrid.DataViewEnum.Form
        Case "GroupBy"
            Me.ClTrueDBGrid1.DataView = Cl.Win.ClTrueDBGrid.DataViewEnum.GroupBy
        Case "MultipleLines"
            Me.ClTrueDBGrid1.DataView =
Cl.Win.ClTrueDBGrid.DataViewEnum.MultipleLines
        Case "Hierarchical"
            MessageBox.Show ("Hierarchical View can't be set at run time. Please
see the Hierarchical Display tutorial")
    End Select
End Sub
```

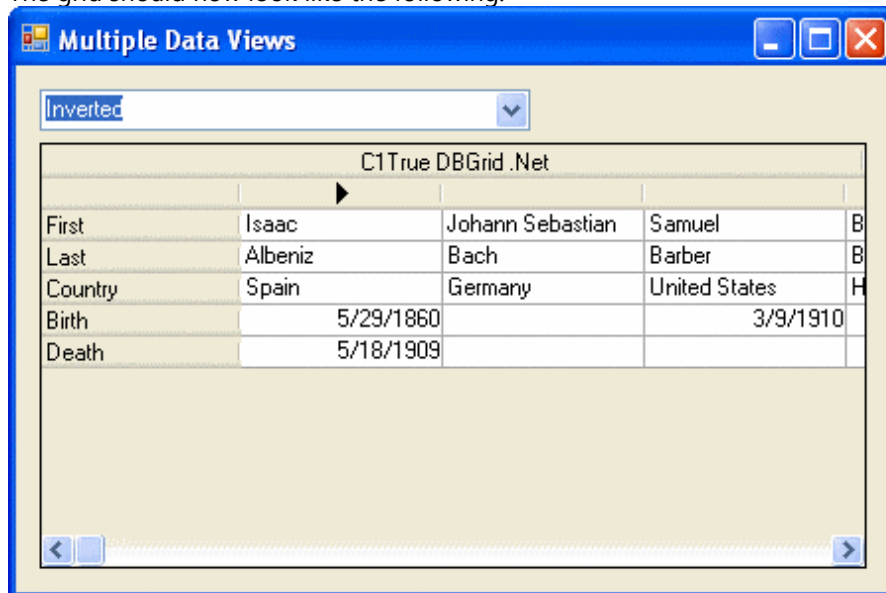
To write code in C#**C#**

```
private void ComboBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    switch (ComboBox1.SelectedItem)
    {
        case "Normal":
            this.clTrueDBGrid1.DataView =
Cl.Win.ClTrueDBGrid.DataViewEnum.Normal;
            break;
        case "Inverted":
            this.clTrueDBGrid1.DataView =
Cl.Win.ClTrueDBGrid.DataViewEnum.Inverted;
            break;
        case "Form":
            this.clTrueDBGrid1.DataView = Cl.Win.ClTrueDBGrid.DataViewEnum.Form;
            break;
        case "GroupBy":
            this.clTrueDBGrid1.DataView =
```

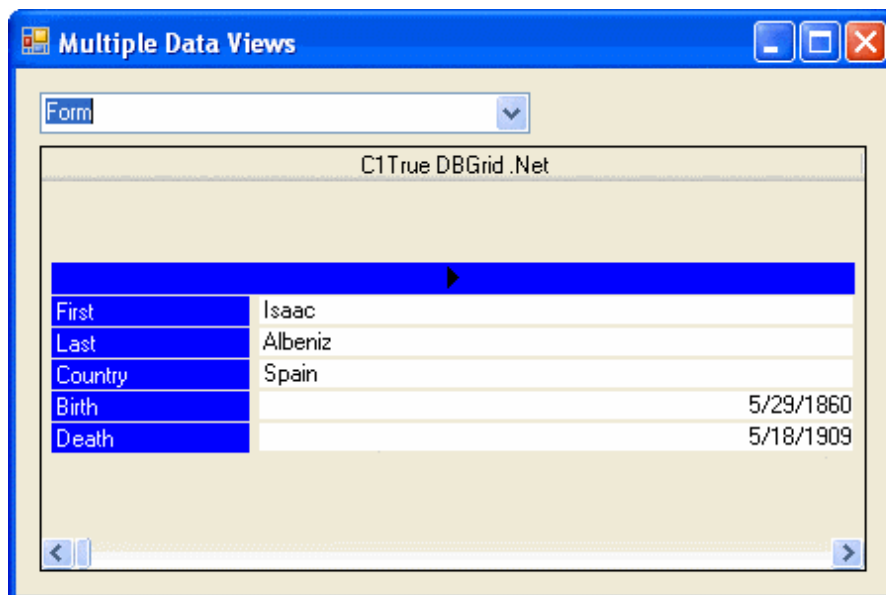
```
C1.Win.C1TrueDBGrid.DataViewEnum.GroupBy;  
    break;  
    case "MultipleLines":  
        this.c1TrueDBGrid1.DataView =  
C1.Win.C1TrueDBGrid.DataViewEnum.MultipleLines;  
        break;  
    case "Hierarchical";  
        MessageBox.Show ("Hierarchical View can't be set at run time. Please  
see the Hierarchical Display tutorial");  
        break;  
    }  
}
```

Run the program and observe the following:

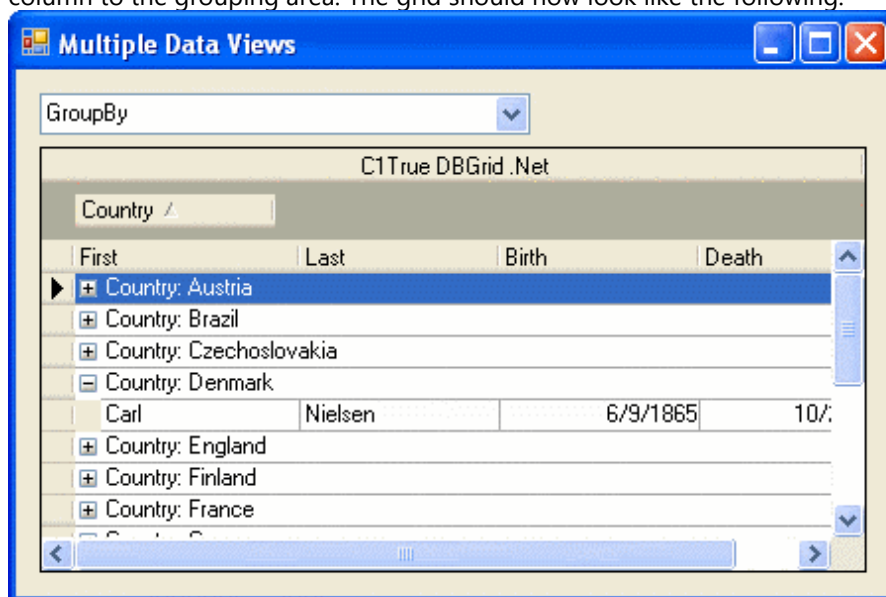
- C1TrueDBGrid1 displays the data specified in [Tutorial 1: Binding True DBGrid to a DataSet](#).
- Change the ComboBox to **Inverted**. Inverted view shows the grid columns as rows and the grid rows as column. The grid should now look like the following:



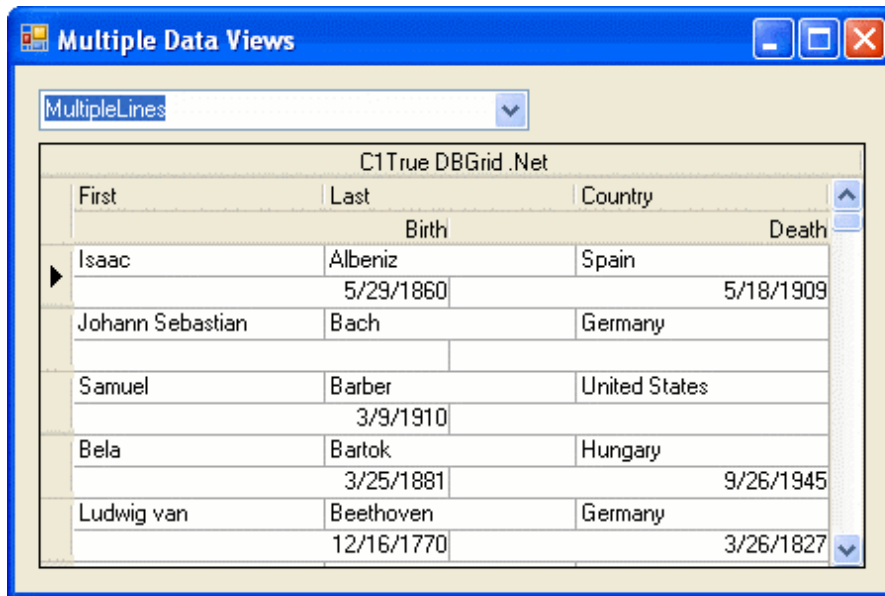
- Change the ComboBox to **Form**. Form view shows each record in a Form-like view that is optimal for data-entry. The grid should now look like the following:



- Change the ComboBox to **GroupBy**. GroupBy View contains a grouping section above the grid where columns can be dragged. Dragging a column to this area sorts the rest of the grid by this column. Drag the Company column to the grouping area. The grid should now look like the following:



- Change the ComboBox to **MultipleLines**. The MultipleLines View shows all of the columns in the current grid area, wrapping the columns that will not fit to successive lines. Notice that the three columns that would have spilled off of the grid are now on a second line. The grid should now look like the following:



- Now set the ComboBox to **Hierarchical**. No changes occur and the Message Box statement included in the event above pops up which is due to the fact that the hierarchical DataView cannot be set at run time. Hierarchical data must be set before the application runs. For more information on this view, see [Tutorial 16: Using the Hierarchical Display](#).

You've successfully completed displaying multiple data views; this concludes the tutorial.

Tutorial 21: Adding a Filter Bar

In this tutorial, you will learn how to use the grid's Filter Bar functionality to allow the end user to sort column data dynamically at run time. Complete the following steps:

1. Start with the project created in [Tutorial 1: Binding True DBGrid to a DataSet](#).
2. After the existing code in the **Load** event of Form1 add the following line:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.FilterBar = True
```

To write code in C#

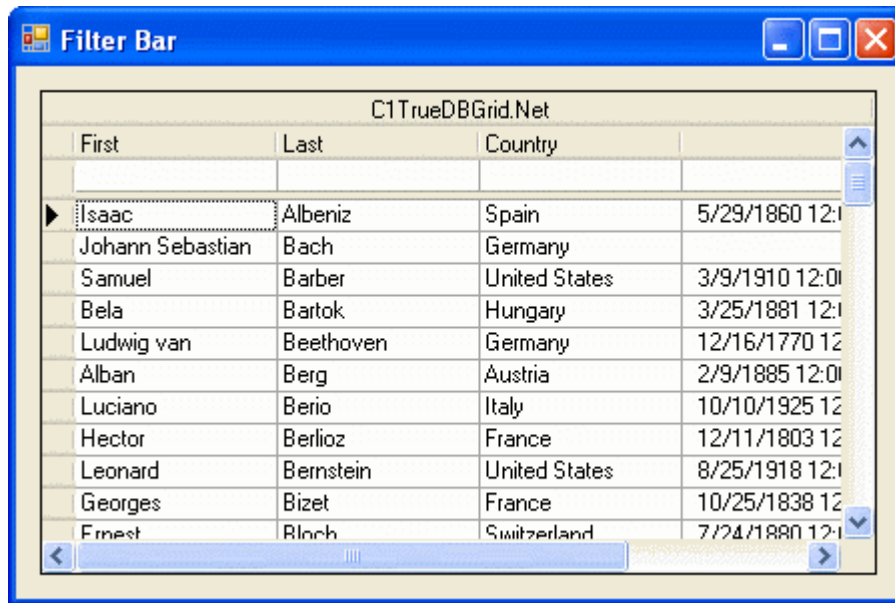
C#

```
this.c1TrueDBGrid1.FilterBar = true;
```

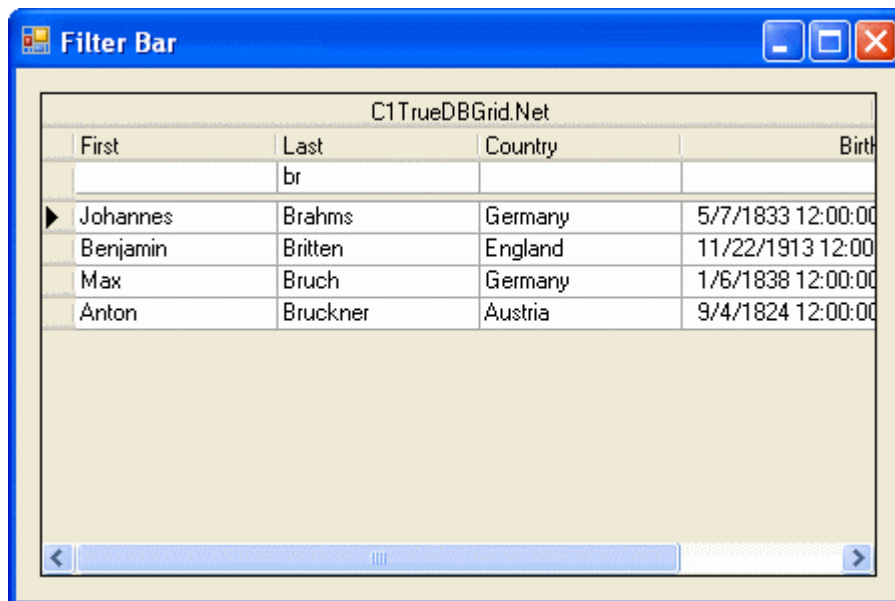
Run the program and observe the following:

- C1TrueDBGrid1 displays the data specified in [Tutorial 1: Binding True DBGrid to a DataSet](#).
- Above the grid data is now a line that accepts user input. This is the Filter Bar. When a user enters data into the bar the grid automatically filters the column data.

Before Filter:



After Filter:



- If you would prefer to handle the filtering yourself, then you would have to change the **AllowFilter** property to **False** (keeping **FilterBar** equal to **True**). Then you would have to handle the **FilterChange** event which fires each time the state of the Filter Bar changes.

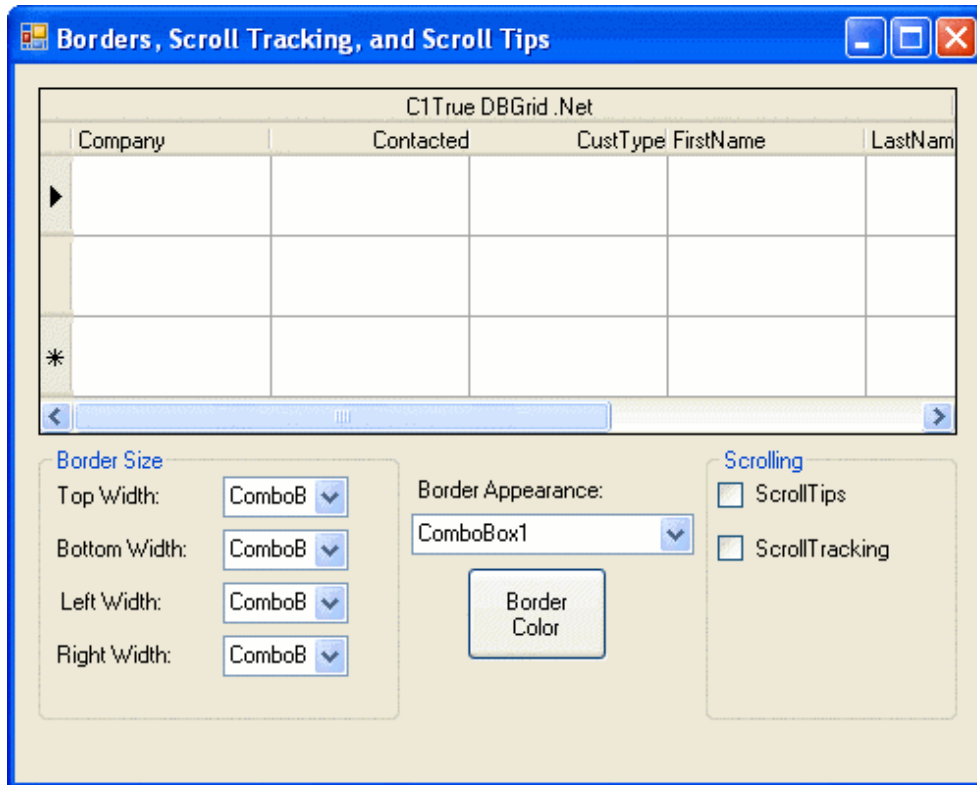
You've successfully completed adding a filter bar; this concludes the tutorial.

Tutorial 22: Borders, Scroll Tracking, and Scroll Tips

In this tutorial, you will learn how to adjust borders, add scroll tracking, and add scroll tips to the grid. Complete the following steps:

1. Create a new project. Add a [C1TrueDBGrid](#) control to the form.
2. Add the following items to the form and situate them like they appear in the following image.

- Five ComboBoxes (ComboBox1 – 5).
- Two GroupBoxes (GroupBox1 – 2) and set their **Text** property to "**Border Size**" and "**Scrolling**" respectively.
- Four Labels (Label1 – 5) and set their Text properties to "**Top Width**", "**Bottom Width**", "**Left Width**", "**Right Width**", and "**Border Appearance**" respectively.
- Button (Button1) and set its **Text** property to "**Border Color**".
- Two Checkboxes and set their text properties to "**ScrollTips**" and "**ScrollTracking**".



3. Add a **Color Dialog** control to the form (ColorDialog1).
4. In the **C1TrueDBGrid Tasks** menu, locate the **Choose Data Source** drop-down and select **Add Project Data Source**. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select all fields in the **Customer** table and type "DsCustomer" into the **DataSet name** box, and then finish out the wizard.
5. Click the grid to give it focus, then in the Properties window set the **RowHeight** property to **40**.
6. Visual Studio adds the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.CustomerTableAdapter.Fill(Me.DsCustomer.Customer)
```

To write code in C#

C#

```
this.CustomerTableAdapter.Fill(this.DsCustomer.Customer);
```

7. In the general section of Form1 add the following declarations:

To write code in Visual Basic

Visual Basic

```
' Copy the data.
Dim dbTable As DataTable

Dim borderColor As Color
Dim borderLeft As Integer, borderTop As Integer, borderRight As Integer,
borderBottom As Integer
Dim borderType As Cl.Win.ClTrueDBGrid.BorderTypeEnum
```

To write code in C#**C#**

```
// Copy the data.
DataTable dbTable;

Color borderColor;
int borderLeft, int borderTop, int borderRight, int borderBottom;
Cl.Win.ClTrueDBGrid.BorderTypeEnum borderType;
```

8. Into the **Load** event of Form1 add the following code:

To write code in Visual Basic**Visual Basic**

```
dbTable = Me.DsCustomer.Tables(0).Copy()

' Fill each combobox.
FillComboBox1()
FillCombo(ComboBox2)
FillCombo(ComboBox3)
FillCombo(ComboBox4)
FillCombo(ComboBox5)
Me.CheckBox2.Checked = True

' Initalize border sizes.
Me.borderBottom = 1
Me.borderLeft = 1
Me.borderRight = 1
Me.borderTop = 1
```

To write code in C#**C#**

```
dbTable = this.DsCustomer.Tables[0].Copy();

// Fill each combobox.
FillComboBox1();
FillCombo(comboBox2);
FillCombo(comboBox3);
FillCombo(comboBox4);
FillCombo(comboBox5);
this.checkBox2.Checked = true;
```

```
// Initalize border sizes.  
this.borderBottom = 1;  
this.borderLeft = 1;  
this.borderRight = 1;  
this.borderTop = 1;
```

9. Now add the functions that will fill the ComboBoxes:

To write code in Visual Basic

Visual Basic

```
' Fill each combo with numbers from 1 to 10.  
Private Sub FillCombo(ByRef com As ComboBox)  
    Dim i As Integer  
    com.Text = 1  
    For i = 1 To 10  
        com.Items.Add(i)  
    Next  
End Sub  
  
' Fill the first combo with border types.  
Private Sub FillComboBox1()  
    Me.ComboBox1.Text = "None"  
    With Me.ComboBox1.Items  
        .Add("Fillet")  
        .Add("Flat")  
        .Add("Groove")  
        .Add("Inset")  
        .Add("InsetBevel")  
        .Add("None")  
        .Add("Raised")  
        .Add("RaisedBevel")  
    End With  
End Sub
```

To write code in C#

C#

```
// Fill each combo with numbers from 1 to 10.  
private void FillCombo(ref ComboBox com)  
{  
    int i;  
    com.Text = 1;  
    for (i = 1 ; i <= 10; i++)  
    {  
        com.Items.Add[i];  
    }  
}  
  
// Fill the first combo with border types.  
private void FillComboBox1()  
{  
    this.comboBox1.Text = "None";  
}
```

```

        this.comboBox1.Items.Add("Fillet");
        this.comboBox1.Items.Add("Flat");
        this.comboBox1.Items.Add("Groove");
        this.comboBox1.Items.Add("Inset");
        this.comboBox1.Items.Add("InsetBevel");
        this.comboBox1.Items.Add("None");
        this.comboBox1.Items.Add("Raised");
        this.comboBox1.Items.Add("RaisedBevel");
    }

```

10. Now create a handler for the **Click** event of Button1 that will set the color of the border using the color dialog box:

To write code in Visual Basic

Visual Basic

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim result As DialogResult
    result = Me.ColorDialog1.ShowDialog()
    If result = DialogResult.OK Then
        borderColor = Me.ColorDialog1.Color
        Button1.BackColor = borderColor
    End If
    UpdateBorder()
End Sub

```

To write code in C#

C#

```

private void button1_Click(System.Object sender, System.EventArgs e)
button1.Click {
    DialogResult result;
    result = this.colorDialog1.ShowDialog();
    if (result == DialogResult.OK )
    {
        borderColor = this.colorDialog1.Color;
        button1.BackColor = borderColor;
    }
    UpdateBorder();
}

```

11. Now include the function that updates the borders:

To write code in Visual Basic

Visual Basic

```

Private Sub UpdateBorder()
    With
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(Me.C1TrueDBGrid1.Col).Style.Borders
        .Color = ColorDialog1.Color
        .BorderType = borderType
        .Bottom = borderBottom
    End With

```

```

        .Left = borderLeft
        .Right = borderRight
        .Top = borderTop
    End With
End Sub

```

To write code in C#

C#

```

private void UpdateBorder()
{
    Cl.Win.ClTrueDBGrid.GridBorders b;
    b =
this.clTrueDBGrid1.Splits[0].DisplayColumns(this.clTrueDBGrid1.Col).Style.Borders;
    b.Color = colorDialog1.Color;
    b.BorderType = borderType;
    b.Bottom = borderBottom;
    b.Left = borderLeft;
    b.Right = borderRight;
    b.Top = borderTop;
}

```

12. Now include the code that handles changes in the **ComboBox** values:

To write code in Visual Basic

Visual Basic

```

Private Sub ComboBox1_SelectionChangeCommitted(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ComboBox1.SelectionChangeCommitted
    Select Case Me.ComboBox1.SelectedItem
        Case "Fillet"
            Me.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.Fillet
        Case "Flat"
            Me.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.Flat
        Case "Groove"
            Me.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.Groove
        Case "Inset"
            Me.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.Inset
        Case "InsetBevel"
            Me.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.InsetBevel
        Case "None"
            Me.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.None
        Case "Raised"
            Me.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.Raised
        Case "RaisedBevel"
            Me.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.RaisedBevel
    End Select
    Me.UpdateBorder()
End Sub

Private Sub ComboBox2_SelectionChangeCommitted(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ComboBox2.SelectionChangeCommitted
    Me.borderTop = Me.ComboBox2.SelectedItem

```

```
        Me.UpdateBorder()  
End Sub  
  
Private Sub ComboBox3_SelectionChangeCommitted(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles ComboBox3.SelectionChangeCommitted  
    Me.borderBottom = Me.ComboBox3.SelectedItem  
    Me.UpdateBorder()  
End Sub  
  
Private Sub ComboBox4_SelectionChangeCommitted(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles ComboBox4.SelectionChangeCommitted  
    Me.borderLeft = Me.ComboBox4.SelectedItem  
    Me.UpdateBorder()  
End Sub  
  
Private Sub ComboBox5_SelectionChangeCommitted(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles ComboBox5.SelectionChangeCommitted  
    Me.borderRight = Me.ComboBox5.SelectedItem  
    Me.UpdateBorder()  
End Sub
```

To write code in C#

C#

```
private void ComboBox1_SelectionChangeCommitted(object sender, System.EventArgs  
e) {  
    switch (this.comboBox1.SelectedItem)  
    {  
        case "Fillet";  
            this.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.Fillet;  
            break;  
        case "Flat";  
            this.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.Flat;  
            break;  
        case "Groove";  
            this.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.Groove;  
            break;  
        case "Inset";  
            this.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.Inset;  
            break;  
        case "InsetBevel";  
            this.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.InsetBevel;  
            break;  
        case "None";  
            this.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.None;  
            break;  
        case "Raised";  
            this.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.Raised;  
            break;  
        case "RaisedBevel";  
            this.borderType = Cl.Win.ClTrueDBGrid.BorderTypeEnum.RaisedBevel;  
            break;  
    }  
}
```

```

    }
    this.UpdateBorder();
}

private void comboBox2_SelectionChangeCommitted(object sender, System.EventArgs
e) {
    this.borderTop = this.comboBox2.SelectedItem;
    this.UpdateBorder();
}

private void comboBox3_SelectionChangeCommitted(object sender, System.EventArgs
e) {
    this.borderBottom = this.comboBox3.SelectedItem;
    this.UpdateBorder();
}

private void comboBox4_SelectionChangeCommitted(object sender, System.EventArgs
e) {
    this.borderLeft = this.comboBox4.SelectedItem;
    this.UpdateBorder();
}

private void comboBox5_SelectionChangeCommitted(object sender, System.EventArgs
e) {
    this.borderRight = this.comboBox5.SelectedItem;
    this.UpdateBorder();
}

```

13. Finally include the code that handles the check boxes and the **FetchScrollTips** event that sets the ToolTip box that displays when the user is scrolling:

To write code in Visual Basic

Visual Basic

```

Private Sub CheckBox1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles CheckBox1.Click
    Me.C1TrueDBGrid1.ScrollTips = Me.CheckBox1.Checked
End Sub

Private Sub CheckBox2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles CheckBox2.Click
    Me.C1TrueDBGrid1.ScrollTrack = Me.CheckBox2.Checked
End Sub

Private Sub C1TrueDBGrid1_FetchScrollTips(ByVal sender As System.Object, ByVal e
As C1.Win.C1TrueDBGrid.FetchScrollTipsEventArgs) Handles
C1TrueDBGrid1.FetchScrollTips

    ' Set the ScrollTip depending on which scroll bar was moved.
    Select Case e.ScrollBar
        Case C1.Win.C1TrueDBGrid.ScrollBarEnum.Horizontal
            e.ScrollTip = Me.C1TrueDBGrid1.Columns(e.ColIndex).Caption
        Case C1.Win.C1TrueDBGrid.ScrollBarEnum.Vertical

```

```

        e.ScrollTip = "Record: " & CStr(e.Row + 1) & " of " &
CStr(Me.dbTable.Rows.Count) & vbCrLf & "Company: " &
Me.dbTable.Rows(e.Row).Item("Company") & vbCrLf & "User code: " &
Me.dbTable.Rows(e.Row).Item("UserCode")
    End Select
    e.ToolTipStyle.ForeColor = Color.Blue
End Sub

```

To write code in C#

```

C#

private void checkBox1_Click(object sender, System.EventArgs e)
{
    this.c1TrueDBGrid1.ScrollTips = this.checkBox1.Checked;
}

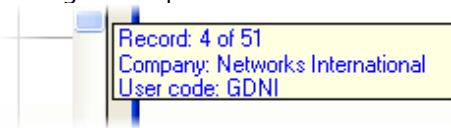
private void checkBox2_Click(object sender, System.EventArgs e)
{
    this.c1TrueDBGrid1.ScrollTrack = this.checkBox2.Checked;
}

private void c1TrueDBGrid1_FetchScrollTips(System.Object sender,
C1.Win.C1TrueDBGrid.FetchScrollTipsEventArgs e)
{
    // Set the ScrollTip depending on which scroll bar was moved.
    switch (e.ScrollBar)
    {
        case C1.Win.C1TrueDBGrid.ScrollBarEnum.Horizontal:
            e.ScrollTip = this.c1TrueDBGrid1.Columns[e.ColIndex].Caption;
            break;
        case C1.Win.C1TrueDBGrid.ScrollBarEnum.Vertical:
            e.ScrollTip = "Record: " + (e.Row + 1).ToString() + " of " +
this.dbTable.Rows.Count.ToString() + "\n" + "Company: " + this.dbTable.Rows[e.Row]
["Company"].ToString() + "\n" + "User code: " + this.dbTable.Rows[e.Row]
["UserCode"].ToString();
            break;
    }
    e.ToolTipStyle.ForeColor = Color.Blue;
}

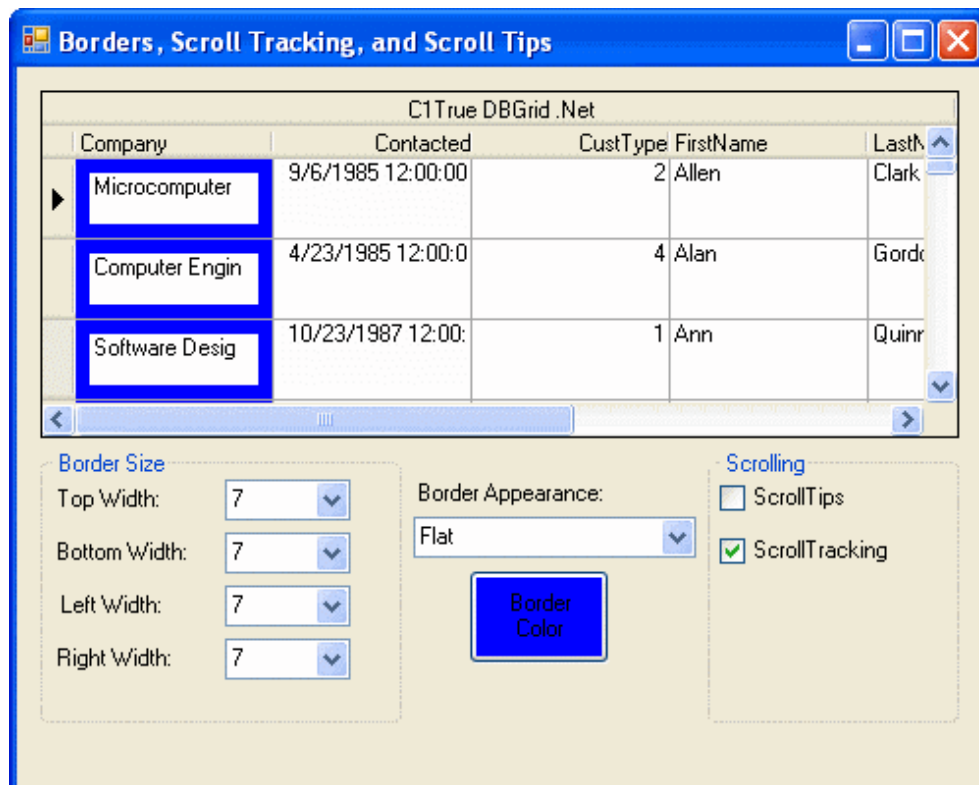
```

Run the program and observe the following:

- C1TrueDBGrid1 displays the data specified.
- Setting ScrollTrack to **True** lets you see the data as it is being scrolled.
- Setting ScrollTips to **True** shows a ToolTip box with column information while the user is scrolling.



- By manipulating the ComboBoxes and the Color Dialog, create a border around a column's cells and set them to a System color.



You've successfully adjusted borders, added scroll tracking, and added scroll tips to the grid; this concludes the tutorial.

True DBGrid for WinForms Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio, and know how to use the [C1TrueDBGrid](#) control in general. If you are unfamiliar with the **True DBGrid for WinForms** product, please see the [True DBGrid for WinForms Tutorials](#) first.

Each topic in this section provides a solution for specific tasks using the **True DBGrid for WinForms** product.

Each task-based help topic also assumes that you have created a new .NET project. Some of the examples reference the **C1NWind.mdb** database which is installed in **Documents\ComponentOne Samples\Common** by default.

Adding a New Row to C1TrueDBGrid

To add a new row to [C1TrueDBGrid](#), use the [AllowAddNew](#) property and the [UpdateData](#) method.

Complete the following steps:

1. Set the **AllowAddNew** property to **True**.

In the Designer

Locate the **AllowAddNew** property in the Properties window and set it to **True**.

In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.AllowAddNew = True
```

To write code in C#

C#

```
this.c1TrueDBGrid1.AllowAddNew = true;
```

2. Move to the last column in the grid by adding the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.MoveLast()
```

To write code in C#

C#

```
this.c1TrueDBGrid.MoveLast();
```

3. Select the new row:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Row = Me.C1TrueDBGrid1.Row + 1
```

```
Me.C1TrueDBGrid1.Select()
```

To write code in C#**C#**

```
this.c1TrueDBGrid1.Row = this.c1TrueDBGrid1.Row + 1;  
this.c1TrueDBGrid1.Select();
```

4. Assign values to the new cells in the first three columns:

To write code in Visual Basic**Visual Basic**

```
Me.C1TrueDBGrid1.Columns(0).Text = "New Row"  
Me.C1TrueDBGrid1.Columns(1).Text = "New Row"  
Me.C1TrueDBGrid1.Columns(2).Text = "New Row"
```

To write code in C#**C#**

```
this.c1TrueDBGrid1.Columns[0].Text = "New Row";  
this.c1TrueDBGrid1.Columns[1].Text = "New Row";  
this.c1TrueDBGrid1.Columns[2].Text = "New Row";
```

5. Update the data to the dataset:

To write code in Visual Basic**Visual Basic**

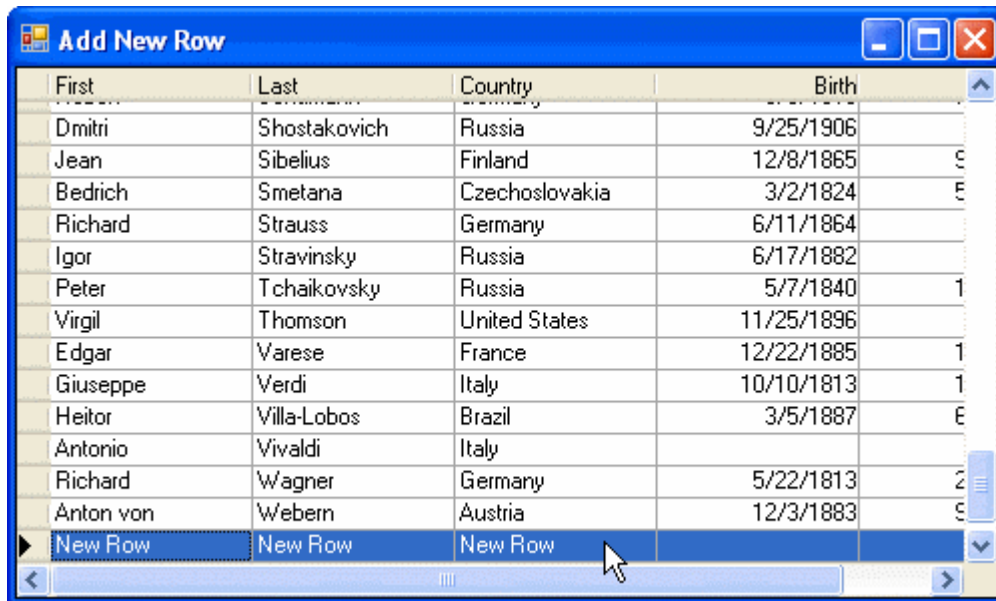
```
Me.C1TrueDBGrid1.UpdateData()
```

To write code in C#**C#**

```
this.c1TrueDBGrid1.UpdateData();
```

What You've Accomplished

In this example, a new row has been added to the C1NWind.mdb:



There is also a [SelectedRows](#) property which points to a collection which contains a reference to all the selected rows in the grid.

Selecting a Row

Highlighting a row does not select the row. In order for the row to be selected, it must be added to the [SelectedRowCollection](#). This can be done using the [Add](#) method.

Add the following code to the **Click** event of the **Select** button:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.SelectedRows.Add(Me.C1TrueDBGrid1.Bookmark)
```

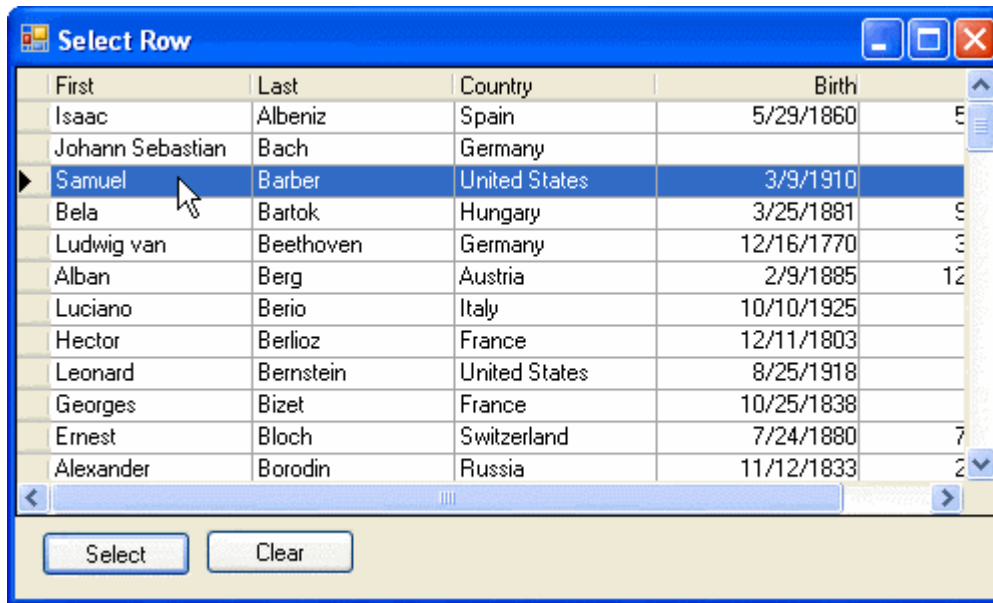
To write code in C#

C#

```
this.c1TrueDBGrid1.SelectedRows.Add(this.c1TrueDBGrid1.Bookmark);
```

What You've Accomplished

Using this example, the current row is selected:



See [Tutorial 5: Selecting Multiple Rows Using Bookmarks](#) for an example of the **Add** method being used.

Accessing the Values of the Selected Rows in the Grid

To access the values of the selected rows in a grid, you must access the **SelectedRows** collection.

Use the following code to write each of the selected rows to the Debug window:

To write code in Visual Basic

Visual Basic

```
Dim row As Integer
For Each row In Me.C1TrueDBGrid1.SelectedRows
    Debug.WriteLine(Me.C1TrueDBGrid1.Columns(0).CellValue(row))
Next
```

To write code in C#

C#

```
int row;
foreach (int row in this.c1TrueDBGrid1.SelectedRows)
{
    Debug.WriteLine(this.c1TrueDBGrid1.Columns(0).CellValue(row));
}
```

You can also use the grid's index to access the rows. Use the following code:

To write code in Visual Basic

Visual Basic

```
Dim row As Integer
For Each row In Me.C1TrueDBGrid1.SelectedRows
    Debug.WriteLine(Me.C1TrueDBGrid1(row, 0).ToString())
```

Next

To write code in C#

C#

```
int row;
foreach (int row in this.c1TrueDBGrid1.SelectedRows)
{
    Debug.WriteLine(this.c1TrueDBGrid1(row, 0).ToString());
}
```

For this example, the following code was added to the **Button1_Click** event in [Tutorial 5: Selecting Multiple Rows Using Bookmarks](#):

To write code in Visual Basic

Visual Basic

```
Dim row As Integer
For Each row In Me.C1TrueDBGrid1.SelectedRows
    Debug.WriteLine(Me.C1TrueDBGrid1(row, 1).ToString())
Next
```

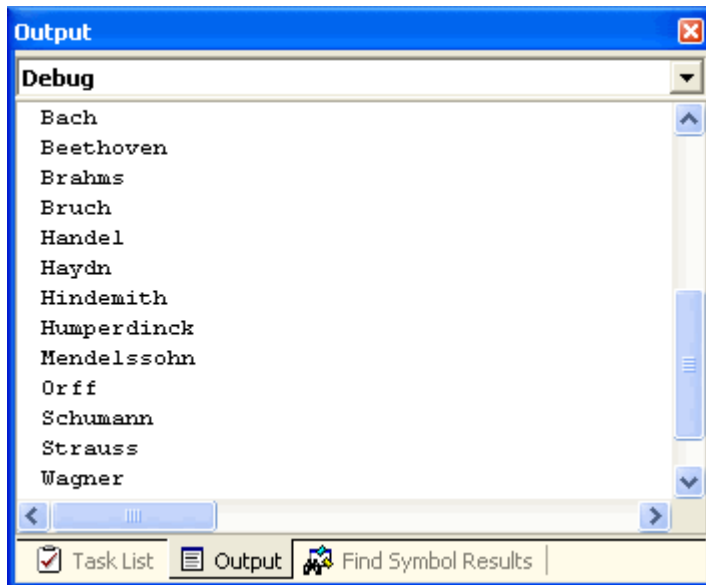
To write code in C#

C#

```
int row;
foreach (int row in this.c1TrueDBGrid1.SelectedRows)
{
    Debug.WriteLine(this.c1TrueDBGrid1(row, 1).ToString());
}
```

What You've Accomplished

The Last name of each composer in a selected row is returned in the Debug window:



Controlling Grid Interaction

The following task-based help topics detail how you can limit your users' interaction with **True DBGrid for WinForms**. For example, you can prevent users from interacting with the grid by sorting, editing, and more.

Disabling Column Sorting

To disable column sorting, set the [AllowSort](#) property to **False**. This property can be set either in the designer or in code.

In the Designer

Locate the [AllowSort](#) property in the Properties window and set it to **False**.

In Code

Add the following code to the **Form_Load** event to set the [AllowSort](#) property to **False**.

To write code in Visual Basic

Visual Basic

```
Me.c1TrueDBGrid1.AllowSort = False
```

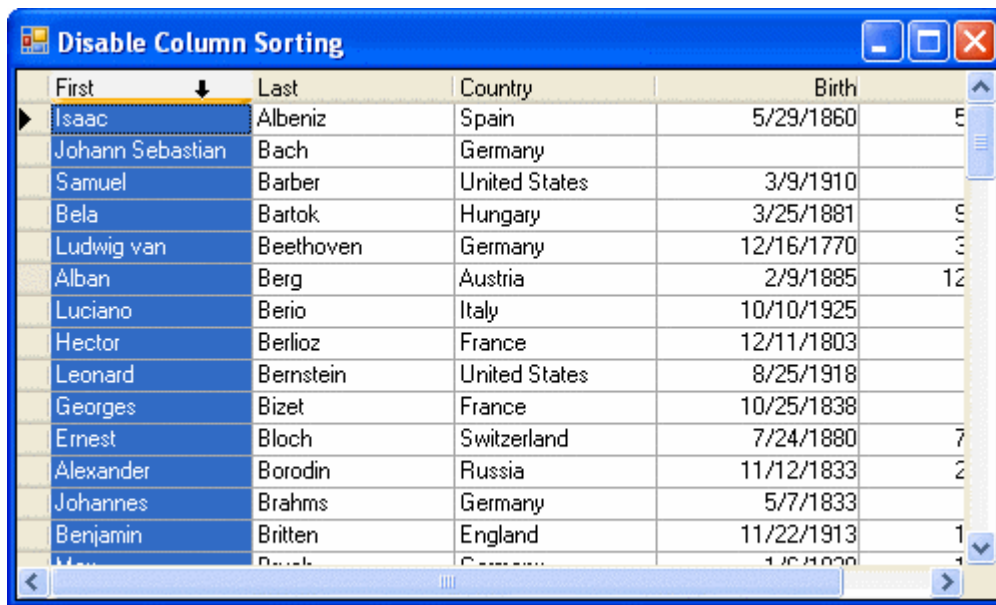
To write code in C#

C#

```
this.c1TrueDBGrid1.AllowSort = false;
```

What You've Accomplished

Clicking on the *First* column does not sort the column:



Locking a Cell from Being Edited

You may want to prevent the end user from editing the data in particular cells. If you choose, you can lock individual grid cells from being edited at run time by using the [FetchCellStyle](#) event.

To lock the value in cell (1, 0), complete the following steps:

1. Set the [FetchStyle](#) property of the column containing the cell to **True**.

In the Designer

- Open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
- Select the *First* column by clicking on its column header in the right pane. Alternatively, it can also be selected from the drop-down list in the toolbar.
- Click the **Display Columns** tab in the left pane.
- Set the [FetchStyle](#) property to **True**.
- Click **OK** to close the designer.

In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(0).FetchStyle = True
```

To write code in C#

```
C#
this.c1TrueDBGrid1.Splits[0].DisplayColumns[0].FetchStyle = true;
```

2. Set the [Locked](#) property of the [CellStyle](#) object to **True** only for the value in row one:

To write code in Visual Basic**Visual Basic**

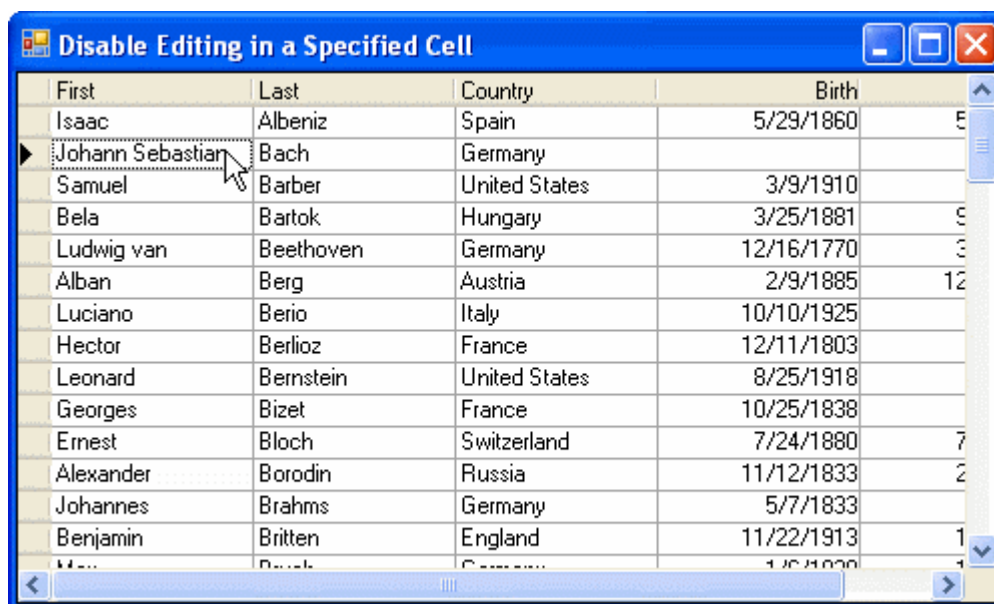
```
Private Sub C1TrueDBGrid1_FetchCellStyle(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.FetchCellStyleEventArgs) Handles
C1TrueDBGrid1.FetchCellStyle
    If e.Row = 1 Then
        e.CellStyle.Locked = True
    End If
End Sub
```

To write code in C#**C#**

```
private void C1TrueDBGrid1_FetchCellStyle(object sender,
C1.Win.C1TrueDBGrid.FetchCellStyleEventArgs e)
{
    if (e.Row == 1)
    {
        e.CellStyle.Locked = true;
    }
}
```

What You've Accomplished

The value in the cell (1, 0) cannot be edited:



Freezing Columns

To freeze columns in the grid, set the **Frozen** property to **True**. Freezing columns locks them from being scrolled and

also prevents all columns with a lesser index from being scrolled. This property can be set either in the designer or in code.

In the Designer

1. Open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
2. In the designer, select the *Last* column by clicking it in the right pane.
The column can also be selected by choosing *Last* from the drop-down list in the toolbar.
3. Click the **Display Column** tab in the left pane.
4. Locate the [Frozen](#) property and set it to **True**.
5. Click **OK** to close the designer.

In Code

Add the following code to the **Form_Load** event to freeze the *Last* column:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns("Last").Frozen = True
```

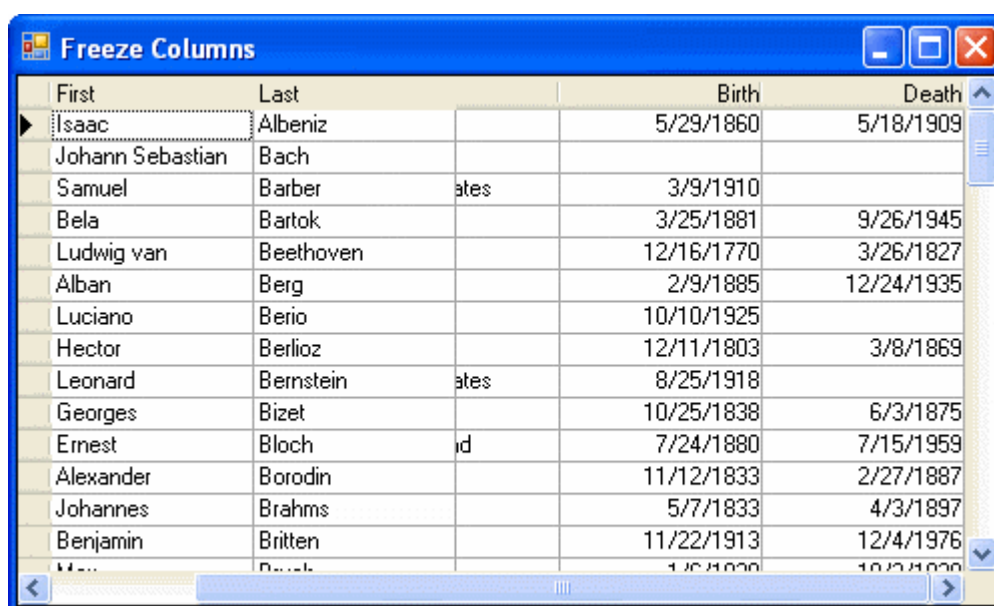
To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].DisplayColumns["Last"].Frozen = true;
```

What You've Accomplished

Both the *First* and *Last* columns are frozen and will remain on the grid when it is scrolled to the right:



Restricting Editing in Specific Columns

To restrict editing in specific columns, set the [Locked](#) property to **True**. This property can be set either in the designer or in code.

In the Designer

Complete the following steps to lock the *Last* column:

1. Open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
2. In the designer, select the *Last* column by clicking it in the right pane.
The column can also be selected by choosing *Last* from the drop-down list in the toolbar.
3. Click the **Display Column** tab in the left pane.
4. Locate the [Locked](#) property and set it to **True**.
5. Click **OK** to close the designer.

In Code

Add the following code to the **Form_Load** event to lock the *Last* column:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns("Last").Locked = True
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].DisplayColumns["Last"].Locked = true;
```

What You've Accomplished

The cells in the *Last* column cannot be edited, but other columns can be edited:



| First | Last | Country | Birth | |
|------------------|-----------|---------------|------------|-----|
| Isaac | Albeniz | Spain | 5/29/1860 | 5 |
| Johann Sebastian | Bach | Germany | | |
| Samuel | Barber | United States | 3/9/1910 | |
| Bela | Bartok | Hungary | 3/25/1881 | 9 |
| Ludwig van | Beethoven | Germany | 12/16/1770 | 3 |
| Alban | Berg | Austria | 2/9/1885 | 12 |
| Luciano | Berio | Italy | 10/10/1925 | |
| Hector | Berlioz | France | 12/11/1803 | |
| Leonard | Bernstein | United States | 8/25/1918 | |
| Georges | Bizet | France | 10/25/1838 | |
| Ernest | Bloch | Switzerland | 7/24/1880 | 7 |
| Alexander | Borodin | Russia | 11/12/1833 | 2 |
| Johannes | Brahms | Germany | 5/7/1833 | |
| Benjamin | Britten | England | 11/22/1913 | 1 |
| ... | ... | ... | ... | ... |

Setting the Grid's Appearance

The following task-based help topics detail how you can change the appearance of **True DBGrid for WinForms** controls. For example, you can add gradients to columns, change the font, set background color and height of rows, and more.

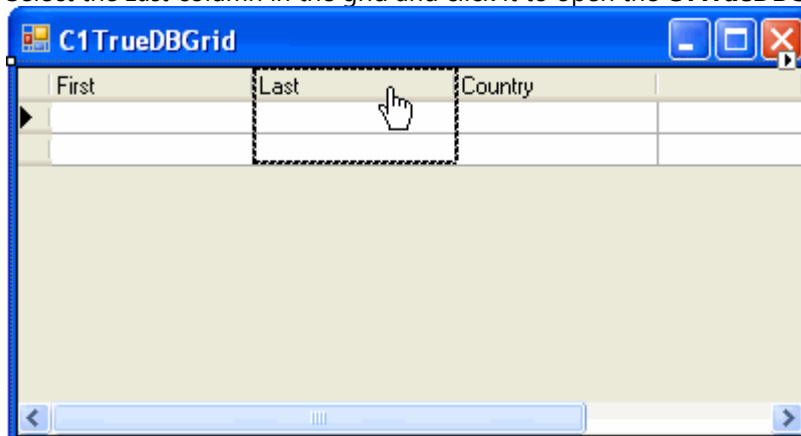
Adding a Gradient Fill to a Column

To add a gradient fill to a column, set the [GradientMode](#), [BackColor](#), and [BackColor2](#) properties. Also, setting the [GammaCorrection](#) property to **True** to apply the gradient with a more uniform intensity. These properties can be set either in the designer or in code.

In the Tasks Menu

Complete the following steps to set the gradient fill using the **C1TrueDBGrid Tasks** menu:

1. Select the *Last* column in the grid and click it to open the **C1TrueDBGrid Tasks** menu.

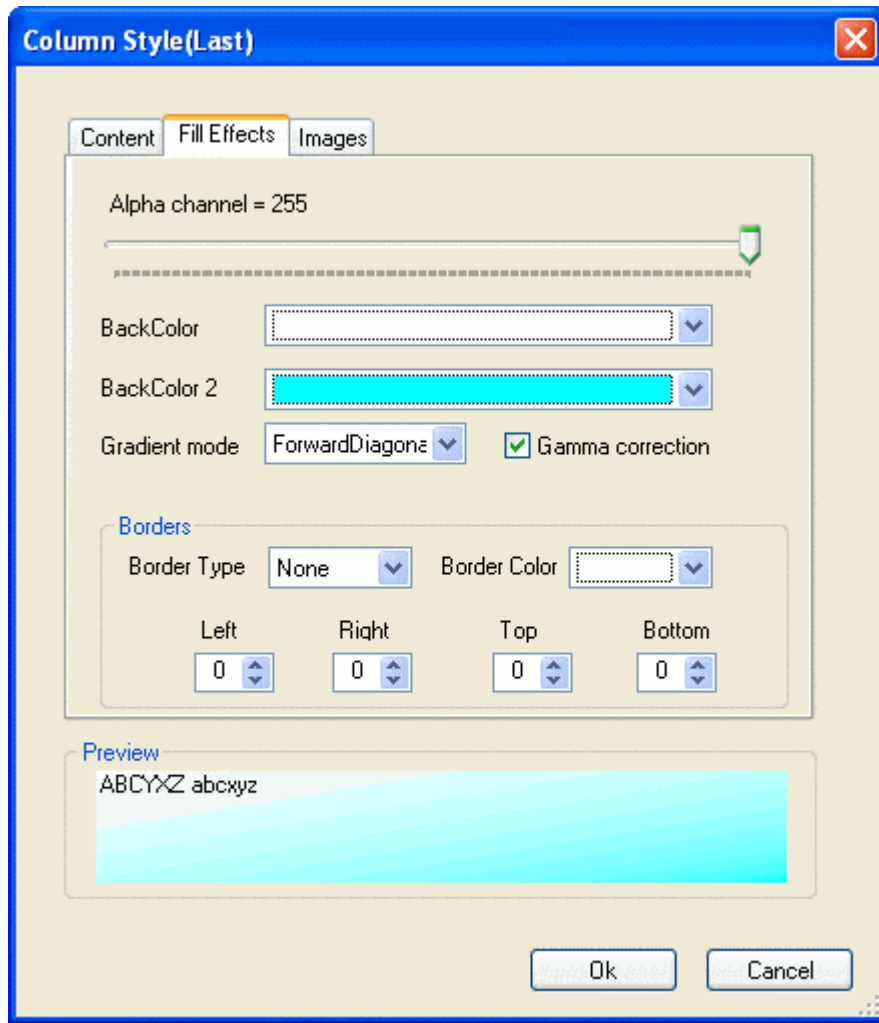


2. Select **Column Style** from the menu.

The screenshot shows the 'C1TrueDBGrid Tasks' dialog box with the 'Column Tasks' tab selected. The dialog has a title bar 'C1TrueDBGrid Tasks' and a tab 'Column Tasks'. It contains several settings:

- Select Column:** A dropdown menu with 'Last' selected.
- Column Caption:** A text box containing 'Last'.
- Data Field:** A dropdown menu with 'Last' selected.
- Input Mask:** An empty text box with a mask icon (three dots in a square) to its right.
- Aggregate:** A dropdown menu with 'None' selected.
- Visible:** A checked checkbox.
- Visible when Grouped:** An unchecked checkbox.
- Edit using DateTimePicker:** A checked checkbox.
- Links:** Four blue text links: 'Caption Style...', 'Column Style...' (with a mouse cursor pointing to it), 'Value Items...', and 'C1TrueDBGrid Tasks'.
- Buttons:** Three blue text buttons: 'Undock in parent container', 'Add Query...', and 'Preview Data...'.

3. Click the **Fill Effects** tab.
4. Set **BackColor 2** to **Aqua**.
5. Set **Gradient mode** to **ForwardDiagonal**.
6. Check the **Gamma correction** box.

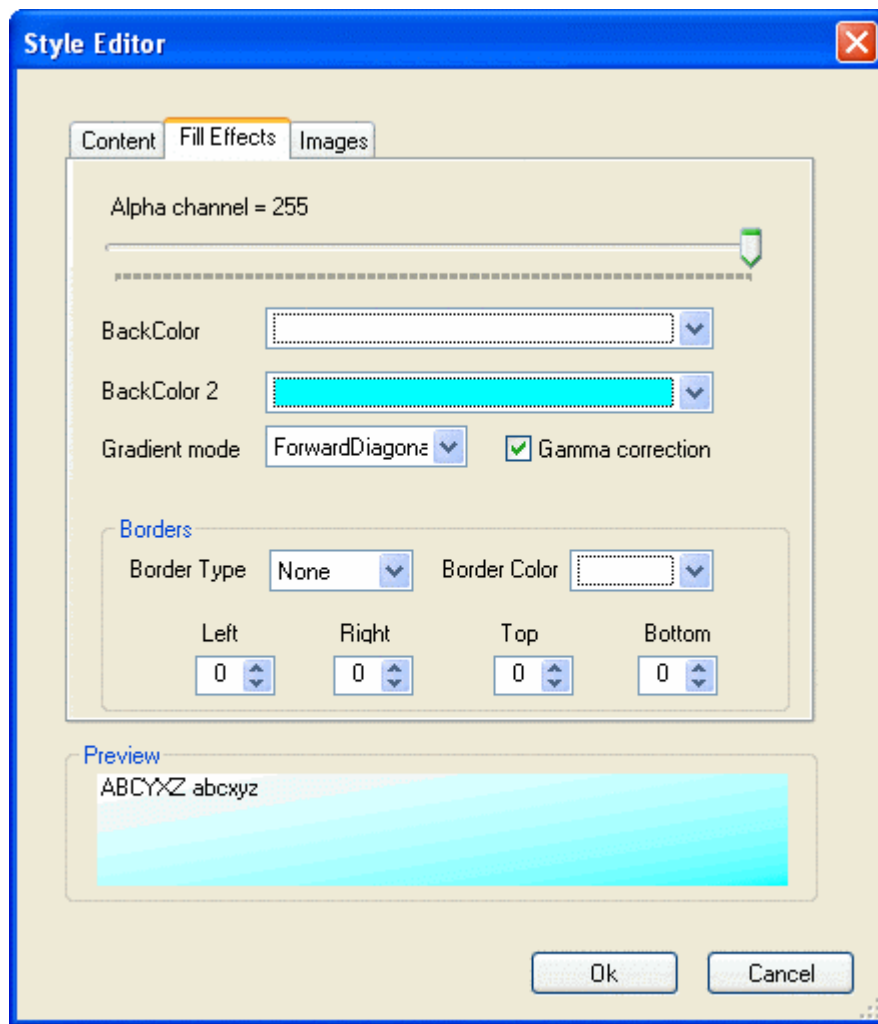


7. Click **Ok** to close the **Column Style(Last)** dialog box.

In the Designer

Alternatively, the gradient fill can also be set through the **C1TrueDBGrid Designer**. To set the gradient fill using the designer, complete the following:

1. Open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
2. Select the *Last* column by clicking it in the right pane.
The column can also be selected by choosing *Last* from the drop-down list in the toolbar.
3. Click the **Display Column** tab in the left pane.
4. Click the **ellipsis** button next to the **Style** property to open the **Style Editor**.
5. In the **Style Editor**, click the **Fill Effects** tab.
6. Set **BackColor 2** to **Aqua**.
7. Set **Gradient mode** to **ForwardDiagonal**.
8. Check the **Gamma correction** box.



9. Click **Ok** to close the **Style Editor**.
10. Click **OK** to close the **C1TrueDBGrid Designer**.

In Code

1. Set **GradientMode** to **ForwardDiagonal** by adding the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns("Last").Style.GradientMode =  
C1.Win.C1TrueDBGrid.GradientModeEnum.ForwardDiagonal
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].DisplayColumns["Last"].Style.GradientMode =  
C1.Win.C1TrueDBGrid.GradientModeEnum.ForwardDiagonal;
```

2. Set **BackColor2** to **Aqua**:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns("Last").Style.BackColor2 = Color.Aqua
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].DisplayColumns["Last"].Style.BackColor2 =  
Color.Aqua;
```

3. Set [GammaCorrection](#) to **True**:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns("Last").Style.GammaCorrection = True
```

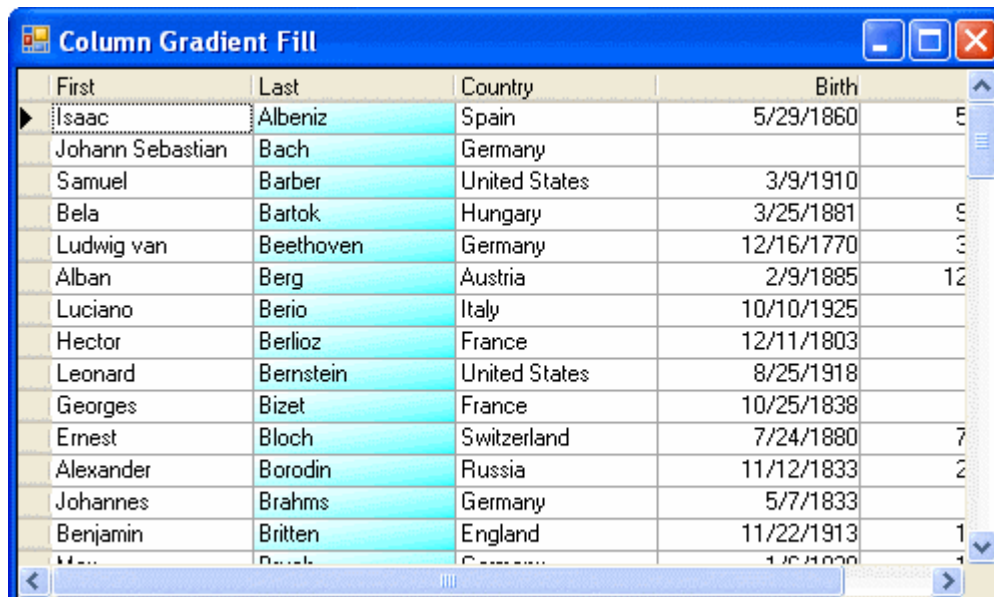
To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].DisplayColumns["Last"].Style.GammaCorrection =  
true;
```

What You've Accomplished

The *Last* column has a white to aqua, forward diagonal gradient fill:



| First | Last | Country | Birth | |
|------------------|-----------|---------------|------------|----|
| Isaac | Albeniz | Spain | 5/29/1860 | 5 |
| Johann Sebastian | Bach | Germany | | |
| Samuel | Barber | United States | 3/9/1910 | |
| Bela | Bartok | Hungary | 3/25/1881 | 9 |
| Ludwig van | Beethoven | Germany | 12/16/1770 | 3 |
| Alban | Berg | Austria | 2/9/1885 | 12 |
| Luciano | Berio | Italy | 10/10/1925 | |
| Hector | Berlioz | France | 12/11/1803 | |
| Leonard | Bernstein | United States | 8/25/1918 | |
| Georges | Bizet | France | 10/25/1838 | |
| Ernest | Bloch | Switzerland | 7/24/1880 | 7 |
| Alexander | Borodin | Russia | 11/12/1833 | 2 |
| Johannes | Brahms | Germany | 5/7/1833 | |
| Benjamin | Britten | England | 11/22/1913 | 1 |
| Max | Buck | Germany | 1/10/1870 | 1 |

Formatting Rows by Specific Criteria

To format rows based on specific criteria, use the [FetchRowStyles](#) property and the [FetchRowStyle](#) event. In this example, rows that do not have values in the *Birth* or *Death* columns will be highlighted green and all other rows will

be locked and formatted in Steel Blue, Tahoma font.

1. Set the [FetchRowStyles](#) property to **True**.

In the Designer

Locate the [FetchRowStyles](#) property in the Properties window and set it to **True**. br/>**In Code**

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.FetchRowStyles = True
```

To write code in C#

C#

```
this.c1TrueDBGrid1.FetchRowStyles = true;
```

2. Add the [FetchRowStyle](#) event:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_FetchRowStyle(ByVal sender As Object, ByVal e As  
C1.Win.C1TrueDBGrid.FetchRowStyleEventArgs) Handles C1TrueDBGrid1.FetchRowStyle  
  
End Sub
```

To write code in C#

C#

```
private void c1TrueDBGrid1_FetchRowStyle(object sender,  
C1.Win.C1TrueDBGrid.FetchRowStyleEventArgs e)  
{  
  
}
```

3. Declare the variables to get the values in the *Birth* and *Death* columns by adding the following code to the [FetchRowStyle](#) event:

To write code in Visual Basic

Visual Basic

```
' Declare variables to get the values in the columns.  
Dim bday As String =  
Me.C1TrueDBGrid1.Columns("Birth").CellText(e.Row).ToString  
Dim ddate As String =  
Me.C1TrueDBGrid1.Columns("Death").CellText(e.Row).ToString
```

To write code in C#

C#

```
// Declare variables to get the values in the columns.
```

```
string bday = this.clTrueDBGrid1.Columns["Birth"].CellText(e.Row).ToString;  
string ddate = this.clTrueDBGrid1.Columns["Death"].CellText(e.Row).ToString;
```

4. Disable editing and change the font if there is an empty cell in either the *Birth* or *Death* column by adding the following code after the code in step 3:

To write code in Visual Basic

Visual Basic

```
' If the Birth or Death column does not contain an empty cell, disable editing  
and change the font.  
If (bday <> "" AndAlso ddate <> "") And (bday <> "" OrElse ddate <> "") Then  
    e.CellStyle.Locked = True  
    e.CellStyle.Font = New Font("Tahoma", 9)  
    e.CellStyle.ForeColor = Color.SteelBlue  
End If
```

To write code in C#

C#

```
// If the Birth or Death column does not contain an empty cell, disable editing  
and change the font.  
if ((bday != "" && ddate != "") And (bday != "" || ddate != ""))  
{  
    e.CellStyle.Locked = true;  
    e.CellStyle.Font = new Font("Tahoma", 9);  
    e.CellStyle.ForeColor = Color.SteelBlue;  
}
```

5. Highlight the rows that contain an empty cell by adding the following code after the code in step 4:

To write code in Visual Basic

Visual Basic

```
' If the Birth or Death column contains an empty cell, highlight the row.  
If bday = "" Or ddate = "" Then  
    e.CellStyle.BackColor = Color.PaleGreen  
End If
```

To write code in C#

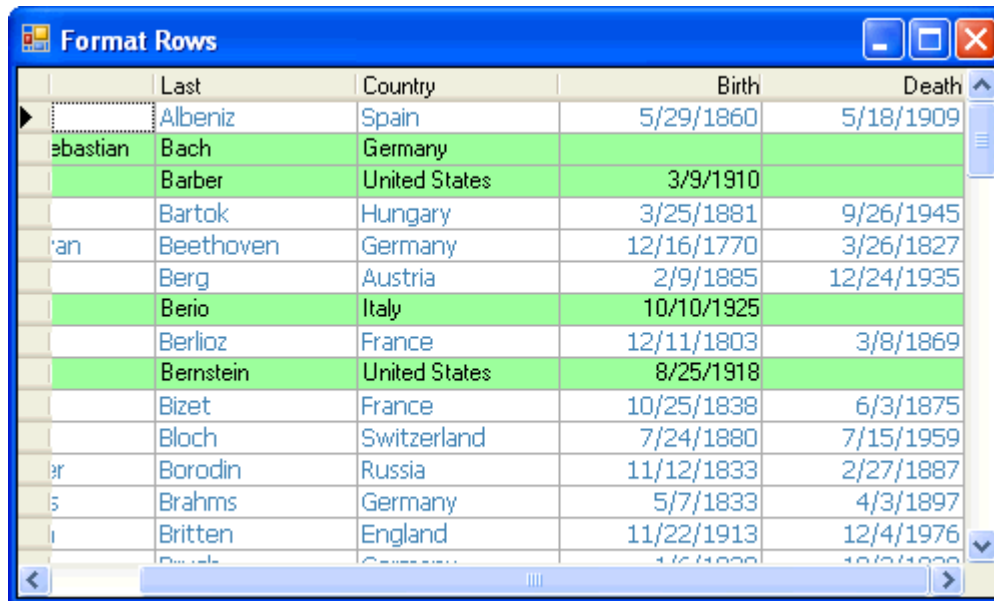
C#

```
// If the Birth or Death column contains an empty cell, highlight the row.  
if (bday == "" || ddate == "")  
{  
    e.CellStyle.BackColor = Color.PaleGreen;  
}
```

What You've Accomplished

Rows with blank values in the *Birth* or *Death* column are highlighted and all other rows are not editable and in a

different font. Adding a value to a blank cell will change the formatting of the cell.



Hiding the Record Selectors Column

The Record Selectors column appears by default at the far left side of the control and it includes an icon to indicate the selected row. To hide the Record Selectors column, set the [RecordSelectors](#) property to **False**. Hiding the Record Selectors column restricts selecting rows. This property can be set either in the designer or in code.

In the Designer

To set the [RecordSelectors](#) property using the **C1TrueDBGrid Designer**:

1. Open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
2. In the designer, click **Record Selectors** in the toolbar to hide the column.



3. Click **OK** to close the **C1TrueDBGrid Designer**.

In the Properties Window

Alternatively, the [RecordSelectors](#) property can also be set in the Properties window. To set the [RecordSelectors](#) property in the Properties window:

- Locate the [RecordSelectors](#) property in the Properties window and set it to **False**.

In Code

Add the following code to the **Form_Load** event to hide the Record Selectors column:

To write code in Visual Basic

Visual Basic

```
Me.clTrueDBGrid1.RecordSelectors = False
```

To write code in C#

C#

```
this.clTrueDBGrid1.RecordSelectors = false;
```

What You've Accomplished

The Record Selectors column is not visible:



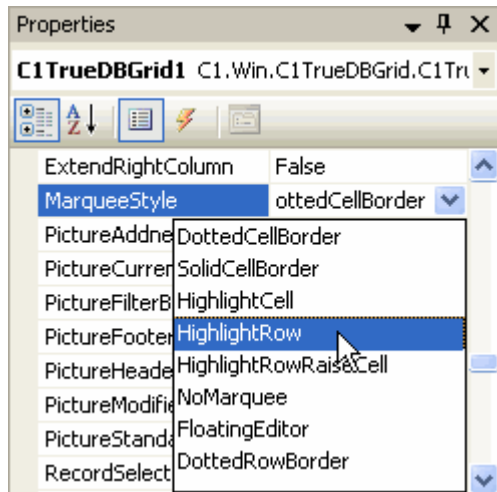
Highlighting the Row of the Selected Cell

To highlight the row of the selected cell, set the [MarqueeStyle](#) property to [HighlightRow](#). This can be set either in the designer or in code.

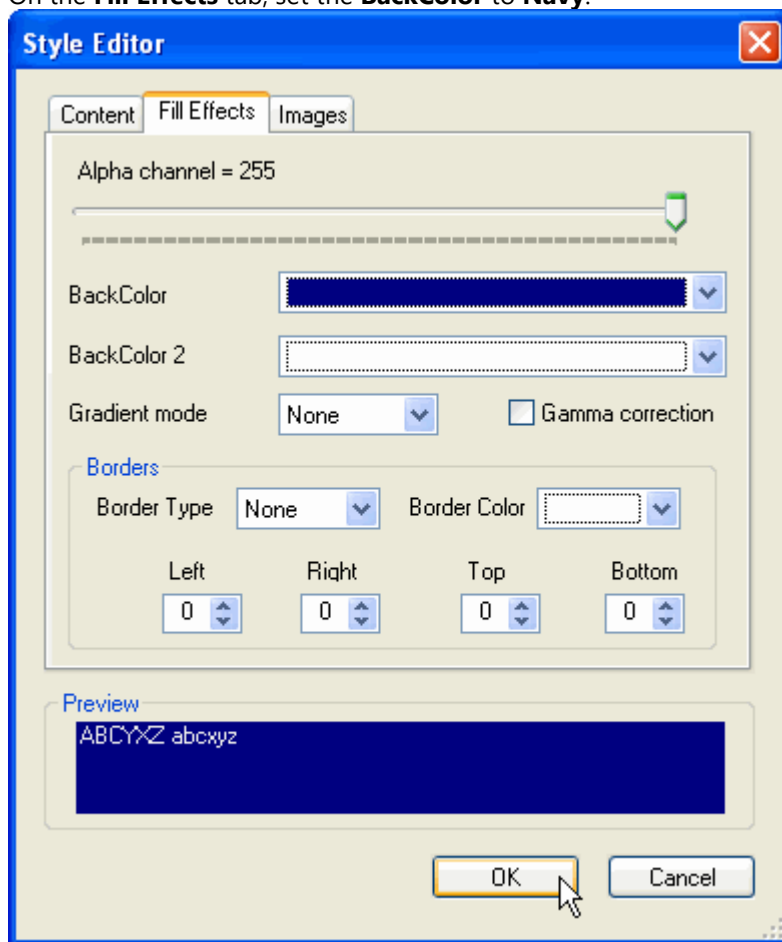
In the Designer

Complete the following steps to highlight the row of the selected cell using the designer:

1. Locate the [MarqueeStyle](#) property in the Properties window and set it to [HighlightRow](#).



2. Click the **ellipsis** button next to the **HighlightRowStyle** property in the Properties window to open the **Style Editor**.
3. On the **Contents** tab, set the **ForeColor** to **WhiteSmoke**.
4. On the **Fill Effects** tab, set the **BackColor** to **Navy**.



5. Click **Ok** to close the **Style Editor**.

In Code

To highlight the row of the selected cell using code, complete the following steps:

1. Set the **MarqueeStyle** property to **HighlightRow** by adding the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.MarqueeStyle = C1.Win.C1TrueDBGrid.MarqueeEnum.HighlightRow
```

To write code in C#

C#

```
this.c1TrueDBGrid1.MarqueeStyle = C1.Win.C1TrueDBGrid.MarqueeEnum.HighlightRow;
```

2. Set the **ForeColor** of the highlight to **WhiteSmoke**:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.HighLightRowStyle.ForeColor = Color.WhiteSmoke
```

To write code in C#

C#

```
this.c1TrueDBGrid1.HighLightRowStyle.ForeColor = Color.WhiteSmoke;
```

3. Set the **BackColor** of the highlight to **Navy**:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.HighLightRowStyle.BackColor = Color.Navy
```

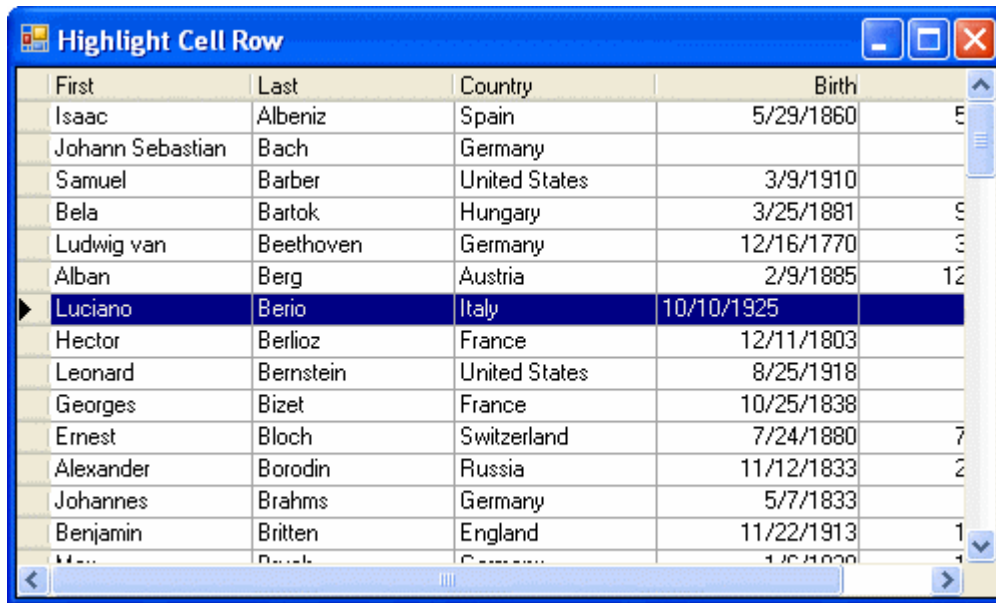
To write code in C#

C#

```
this.c1TrueDBGrid1.HighLightRowStyle.BackColor = Color.Navy;
```

What You've Accomplished

When a cell is selected, the entire row will highlight with the text in WhiteSmoke and the highlight in Navy:



| First | Last | Country | Birth | |
|------------------|-----------|---------------|------------|-----|
| Isaac | Albeniz | Spain | 5/29/1860 | 5 |
| Johann Sebastian | Bach | Germany | | |
| Samuel | Barber | United States | 3/9/1910 | |
| Bela | Bartok | Hungary | 3/25/1881 | 9 |
| Ludwig van | Beethoven | Germany | 12/16/1770 | 3 |
| Alban | Berg | Austria | 2/9/1885 | 12 |
| Luciano | Berio | Italy | 10/10/1925 | |
| Hector | Berlioz | France | 12/11/1803 | |
| Leonard | Bernstein | United States | 8/25/1918 | |
| Georges | Bizet | France | 10/25/1838 | |
| Ernest | Bloch | Switzerland | 7/24/1880 | 7 |
| Alexander | Borodin | Russia | 11/12/1833 | 2 |
| Johannes | Brahms | Germany | 5/7/1833 | |
| Benjamin | Britten | England | 11/22/1913 | 1 |
| ... | ... | ... | ... | ... |

Disabling Selected Highlight

You can disable cell highlighting by setting the [MarqueeStyle](#) and [SelectedStyle](#) properties. To highlight the row of the selected cell, set the [MarqueeStyle](#) property to [NoMarquee](#). See [Highlighting the Current Row or Cell](#) for details. To disable the selected style, change the [SelectedStyle](#) property's attributes.

In the Designer

Complete the following steps to highlight the row of the selected cell using the designer:

1. Locate the [MarqueeStyle](#) property in the Properties window and set it to [NoMarquee](#).
2. Click the **ellipsis** button next to the **SelectedStyle** property in the Properties window to open the **Style Editor**.
3. On the **Contents** tab, set the **ForeColor** to **Black**.
4. On the **Fill Effects** tab, set the **BackColor** to **Transparent**.
5. Click **Ok** to close the **Style Editor**.

In Code

To highlight the row of the selected cell using code, complete the following steps:

1. Set the [MarqueeStyle](#) property to [NoMarquee](#) by adding the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.MarqueeStyle = C1.Win.C1TrueDBGrid.MarqueeEnum.NoMarquee
```

To write code in C#

C#

```
this.c1TrueDBGrid1.MarqueeStyle = C1.Win.C1TrueDBGrid.MarqueeEnum.NoMarquee;
```

2. Set the **ForeColor** of the highlight to **Black**:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.SelectedStyle.ForeColor = Color.Black
```

To write code in C#

C#

```
this.c1TrueDBGrid1.SelectedStyle.ForeColor = Color.Black;
```

3. Set the **BackColor** of the highlight to **Transparent**:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Selected.BackColor = Color.Transparent
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Selected.BackColor = Color.Transparent;
```

What You've Accomplished

Selected cells, rows, and columns will no longer display any indicating highlighting.

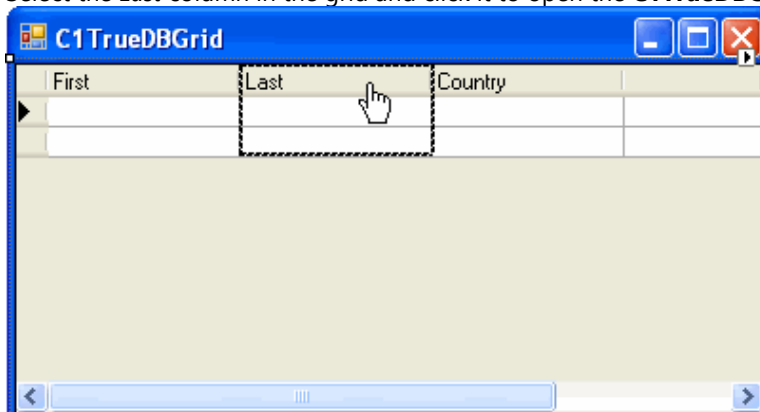
Placing an Image in a Column Header

To place an image in a column header, set the [ForegroundImage](#) and [ForegroundPicturePosition](#) properties. These properties can be set either in the designer or in code.

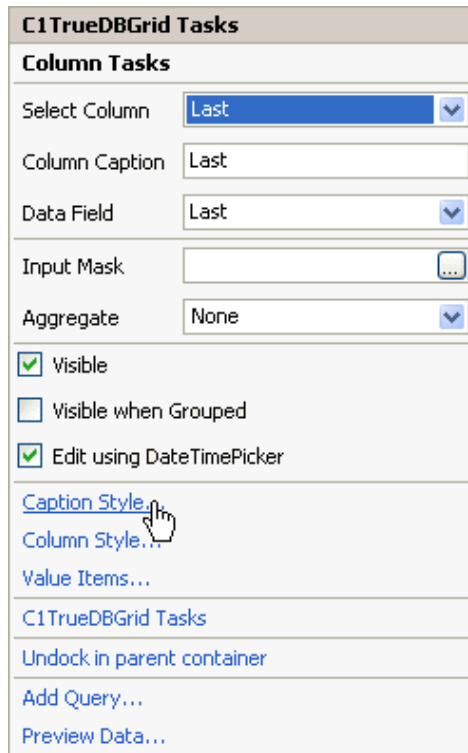
In the Tasks Menu

To place an image in a column header using the **C1TrueDBGrid Tasks** menu:

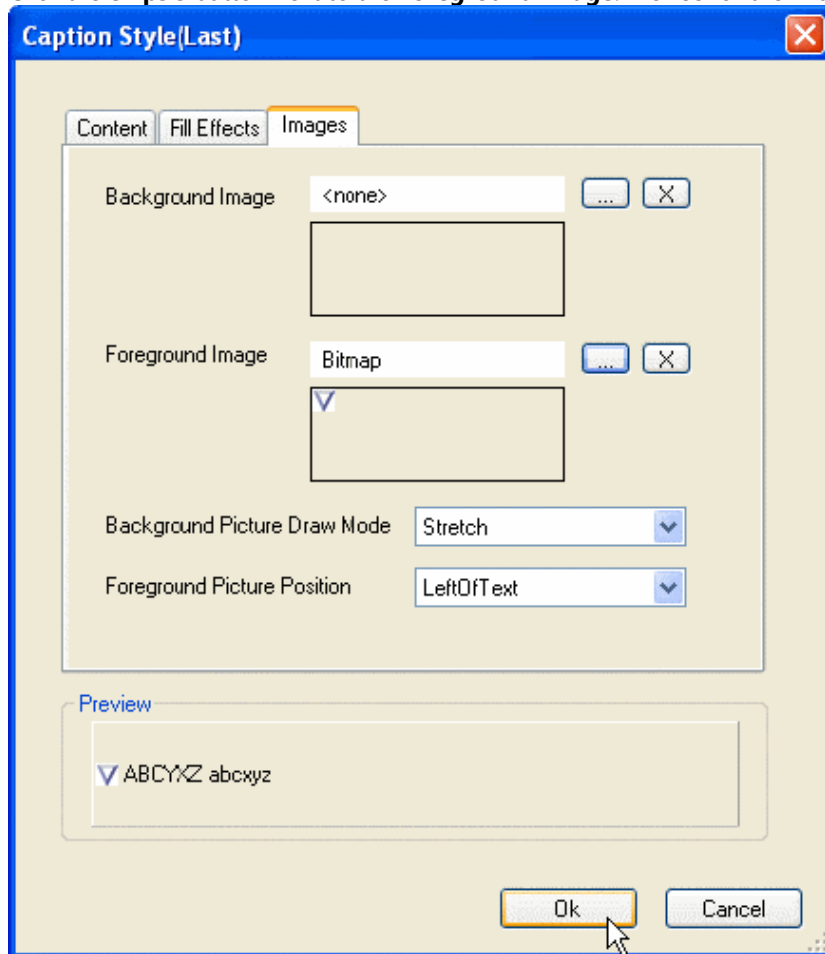
1. Select the *Last* column in the grid and click it to open the **C1TrueDBGrid Tasks** menu.



2. Select **Caption Style** from the menu.



3. Click the **Images** tab.
4. Click the **ellipsis** button next to the **Foreground Image**. Browse for the image and click **Open**.

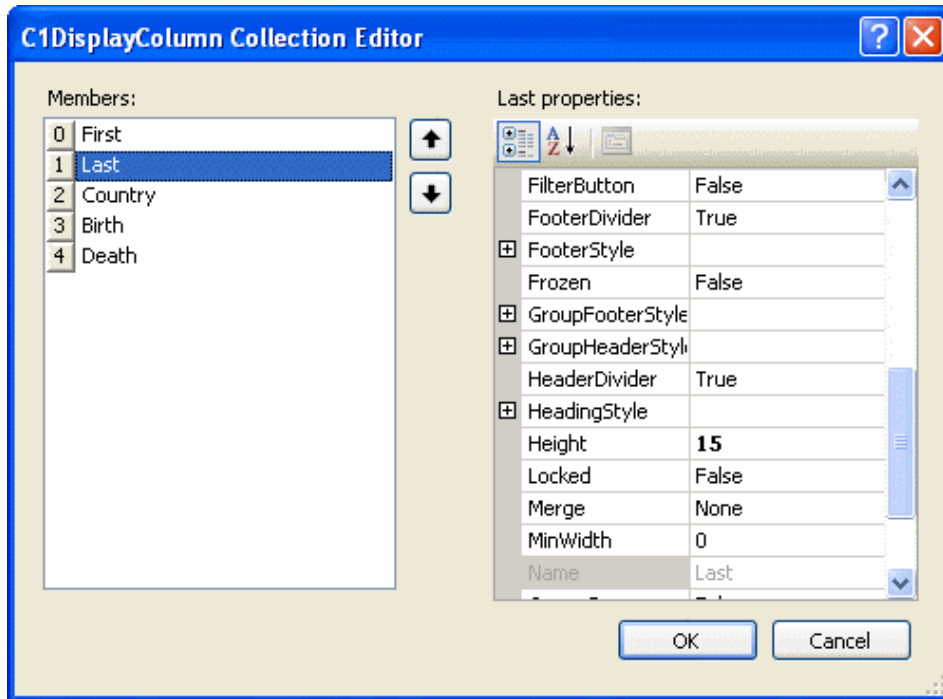


5. Specify the position of the picture using the **Foreground Picture Position** drop-down box.
6. Click **Ok** to close the **Caption Style(Last)** dialog box.

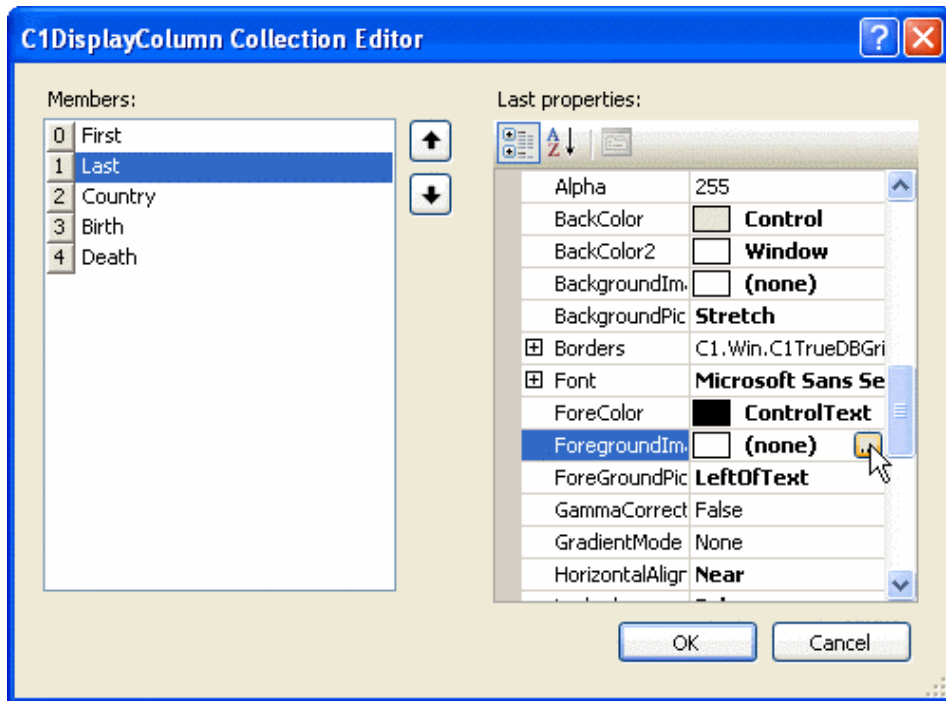
In the C1DisplayColumn Collection Editor

Alternatively, an image can also be placed in the column header at design time using the **C1DisplayColumn Collection Editor**.

1. Click the **ellipsis** button (...) next to the **Splits** property of the grid in the Visual Studio Properties window. The **Split Collection Editor** appears.
2. Click the **ellipsis** button next to the **DisplayColumns** property to bring up the **C1DisplayColumn Collection Editor**.
3. Select the column header from the list of **Members** on the left-hand side.



4. Expand the **HeadingStyle** property on the right-hand side.
5. Click the **ellipsis** button next to the **ForegroundImage** property.



6. Browse for a graphic and click **Open**.
7. Specify the position of the graphic using the [ForegroundPicturePosition](#) property.
8. Click **OK** to close the **C1DisplayColumn Collection Editor**.
9. Click **OK** again to close the **Split Collection Editor**.

In Code

Add the following code to the **Form_Load** event:

1. Specify the image to appear in the header:

To write code in Visual Basic

Visual Basic

```
Dim bmp As New Bitmap("c:\sort.bmp")
```

To write code in C#

C#

```
Bitmap bmp = new Bitmap("c:\\sort.bmp");
```

2. Specify where the image should appear:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(1).HeadingStyle.ForegroundImage = bmp
Me.C1TrueDBGrid1.Splits(0).DisplayColumns(1).HeadingStyle.ForegroundPicturePosition =
C1.Win.C1TrueDBGrid.ForegroundPicturePositionEnum.LeftOfText
```

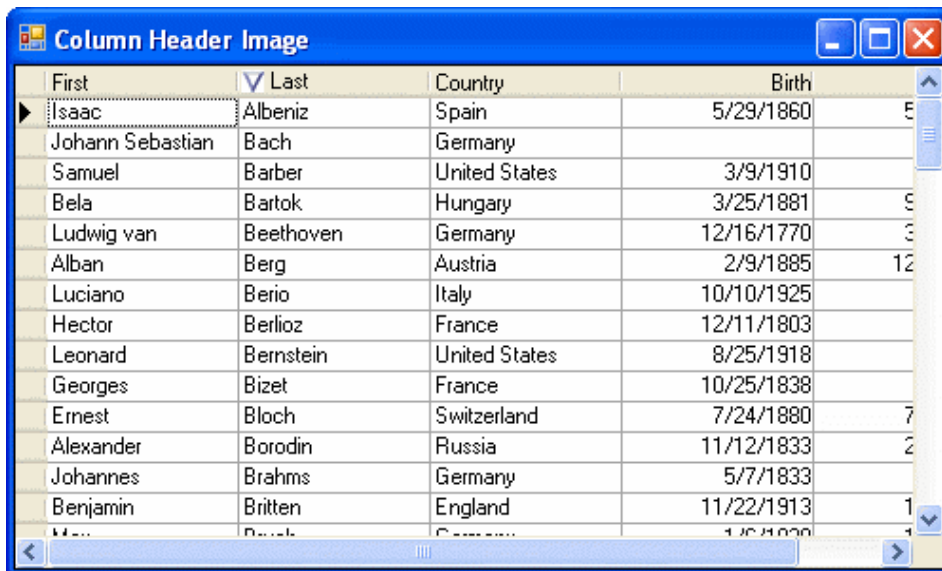
To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].DisplayColumns[1].HeadingStyle.ForegroundImage = bmp;  
this.c1TrueDBGrid1.Splits[0].DisplayColumns[1].HeadingStyle.ForeGroundPicturePosition  
= C1.Win.C1TrueDBGrid.ForeGroundPicturePositionEnum.LeftOfText;
```

What You've Accomplished

In this example, the image appears to the left of the text in the header of the *Last* column:



Setting Multiple Height Values for Rows

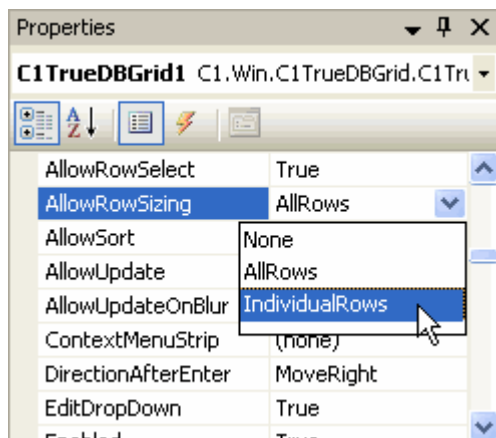
To set multiple height values for rows, set the [AllowRowSizing](#) property to [IndividualRows](#) then assign height values to rows.

Complete the following steps:

1. Set the [AllowRowSizing](#) property to [IndividualRows](#).

In the Designer

Locate the [AllowRowSizing](#) property in the Properties window and set it to [IndividualRows](#).



In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.AllowRowSizing =  
C1.Win.C1TrueDBGrid.RowSizingEnum.IndividualRows
```

To write code in C#

C#

```
this.c1TrueDBGrid1.AllowRowSizing =  
C1.Win.C1TrueDBGrid.RowSizing.IndividualRows;
```

2. Assign different row heights to the first two rows by adding the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).Rows(0).Height = 25  
Me.C1TrueDBGrid1.Splits(0).Rows(1).Height = 50
```

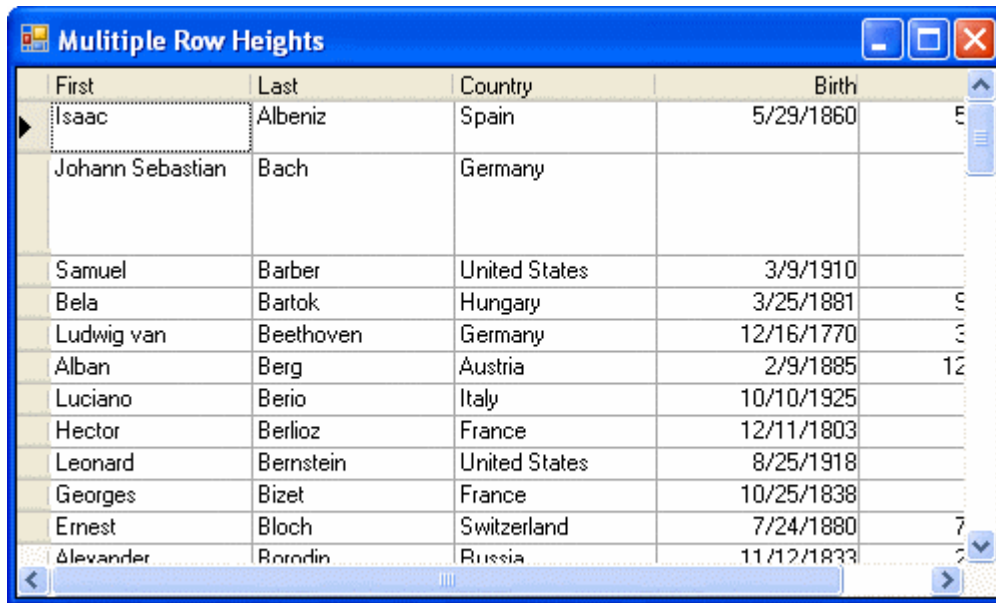
To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].Rows[0].Height = 25;  
this.c1TrueDBGrid1.Splits[0].Rows[1].Height = 50;
```

What You've Accomplished

The first row is set to 25 and the second row is set to 50:



Setting the Background Color of a Row

To set the background color of a row, set the [FetchRowStyles](#) property to fire the [FetchRowStyle](#) event.

Complete the following steps:

1. Set the [FetchRowStyles](#) property to **True**.

In the Designer

Locate the [FetchRowStyles](#) property in the Properties window and set it to **True**.

In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic
Me.C1TrueDBGrid1.FetchRowStyles = True
```

To write code in C#

```
C#
this.c1TrueDBGrid1.FetchRowStyles = true;
```

2. Specify the background color of the desired rows using the [FetchRowStyle](#) event:

To write code in Visual Basic

```
Visual Basic
Private Sub C1TrueDBGrid1_FetchRowStyle(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.FetchRowStyleEventArgs) Handles C1TrueDBGrid1.FetchRowStyle
    Dim S As String = C1TrueDBGrid1.Columns("Country").CellText(e.Row).ToString
    If S <> "Germany" Then
        e.CellStyle.BackColor = System.Drawing.Color.LemonChiffon
```

```
End If
End Sub
```

To write code in C#

```
C#
private void c1TrueDBGrid1_FetchRowStyle(object sender,
C1.Win.C1TrueDBGrid.FetchRowStyleEventArgs e)
{
    string S = c1TrueDBGrid1.Columns("Country").CellText(e.Row).ToString;
    if (S != "Germany")
    {
        e.CellStyle.BackColor = System.Drawing.Color.LemonChiffon;
    }
}
```

What You've Accomplished

In this example, each row that does not contain the word "Germany" in the *Country* column has a background color of lemon chiffon:

| First | Last | Country | Birth | |
|------------------|-----------|---------------|------------|----|
| Isaac | Albeniz | Spain | 5/29/1860 | 5 |
| Johann Sebastian | Bach | Germany | | |
| Samuel | Barber | United States | 3/9/1910 | |
| Bela | Bartok | Hungary | 3/25/1881 | 5 |
| Ludwig van | Beethoven | Germany | 12/16/1770 | 3 |
| Alban | Berg | Austria | 2/9/1885 | 12 |
| Luciano | Berio | Italy | 10/10/1925 | |
| Hector | Berlioz | France | 12/11/1803 | |
| Leonard | Bernstein | United States | 8/25/1918 | |
| Georges | Bizet | France | 10/25/1838 | |
| Ernest | Bloch | Switzerland | 7/24/1880 | 7 |
| Alexander | Borodin | Russia | 11/12/1833 | 2 |
| Johannes | Brahms | Germany | 5/7/1833 | |
| Benjamin | Britten | England | 11/22/1913 | 1 |
| Max | Buck | Germany | 1/10/1870 | 1 |

Setting the Column's Caption Height

To set the column's caption height, set the [ColumnCaptionHeight](#) property. This can be set either in the designer or in code.

In the Designer

To set the column's caption height in the Designer, complete the following steps:

1. Open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
2. Click the **Split** tab in the left pane.
3. Locate the [ColumnCaptionHeight](#) property and set it to **34**.
4. Click the **Column** tab.
5. Locate the [Caption](#) property and change it from **First** to "Composer's First Name".
6. Click **OK** to close the **C1TrueDBGrid Designer**.

In Code

To set the column's caption height in code, complete the following steps:

1. Set the [ColumnCaptionHeight](#) property to fit two rows by adding the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).ColumnCaptionHeight =  
Me.C1TrueDBGrid1.Splits(0).ColumnCaptionHeight * 2
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].ColumnCaptionHeight =  
this.c1TrueDBGrid1.Splits[0].ColumnCaptionHeight * 2;
```

2. Set the [Caption](#) property:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns(0).Caption = "Composer's First Name"
```

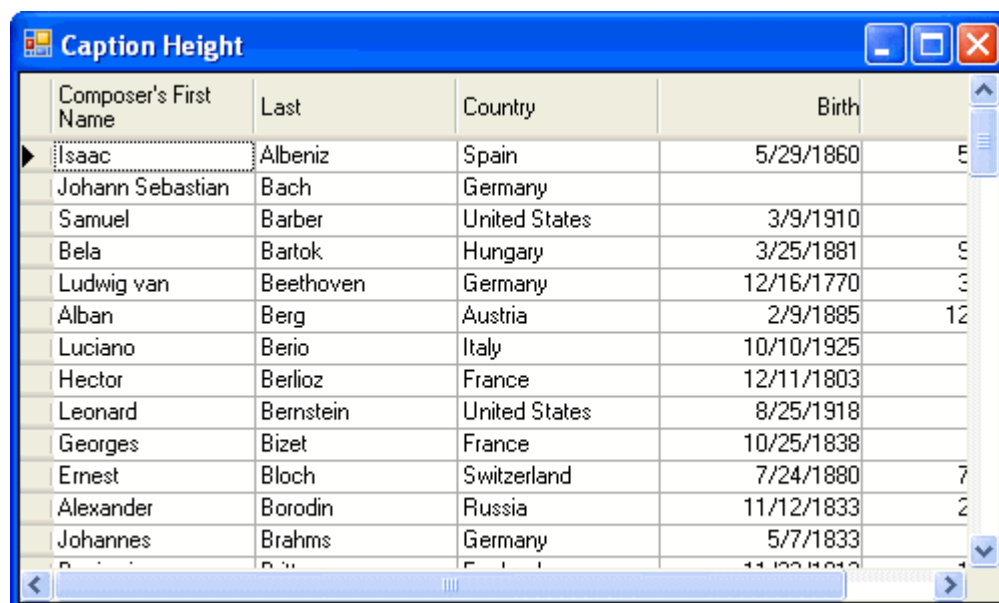
To write code in C#

C#

```
this.c1TrueDBGrid1.Columns[0].Caption = "Composer's First Name";
```

What You've Accomplished

The caption in the column containing first names is set to *Composer's First Name* and spans two rows:



The screenshot shows a TrueDBGrid window titled "Caption Height". It contains a table with the following data:

| Composer's First Name | Last | Country | Birth |
|-----------------------|-----------|---------------|------------|
| Isaac | Albeniz | Spain | 5/29/1860 |
| Johann Sebastian | Bach | Germany | |
| Samuel | Barber | United States | 3/9/1910 |
| Bela | Bartok | Hungary | 3/25/1881 |
| Ludwig van | Beethoven | Germany | 12/16/1770 |
| Alban | Berg | Austria | 2/9/1885 |
| Luciano | Berio | Italy | 10/10/1925 |
| Hector | Berlioz | France | 12/11/1803 |
| Leonard | Bernstein | United States | 8/25/1918 |
| Georges | Bizet | France | 10/25/1838 |
| Ernest | Bloch | Switzerland | 7/24/1880 |
| Alexander | Borodin | Russia | 11/12/1833 |
| Johannes | Brahms | Germany | 5/7/1833 |

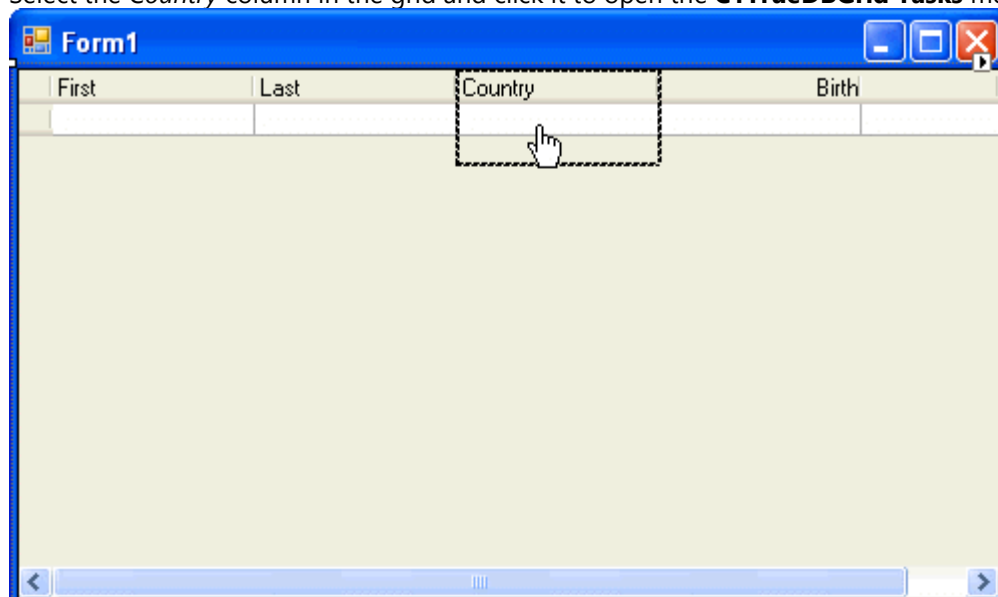
Setting the Font Style of a Column

The font style of a column can be set either in the designer or in code.

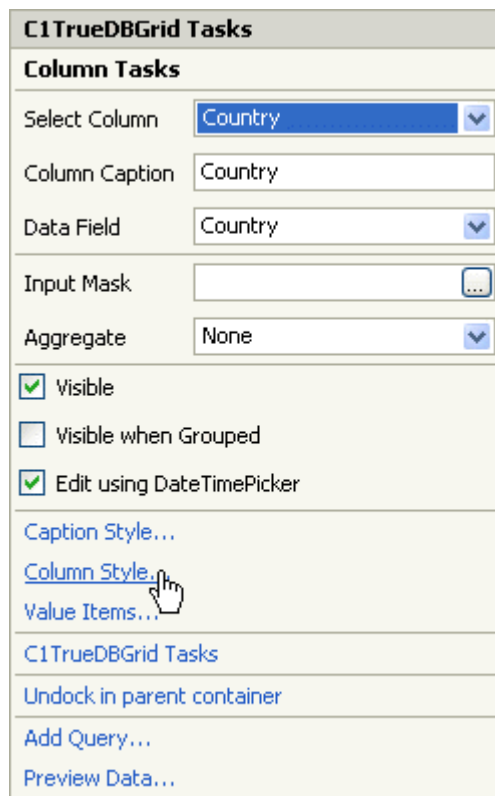
In the Tasks Menu

To set the font style using the **C1TrueDBGrid Tasks** menu, complete the following steps:

1. Select the *Country* column in the grid and click it to open the **C1TrueDBGrid Tasks** menu.



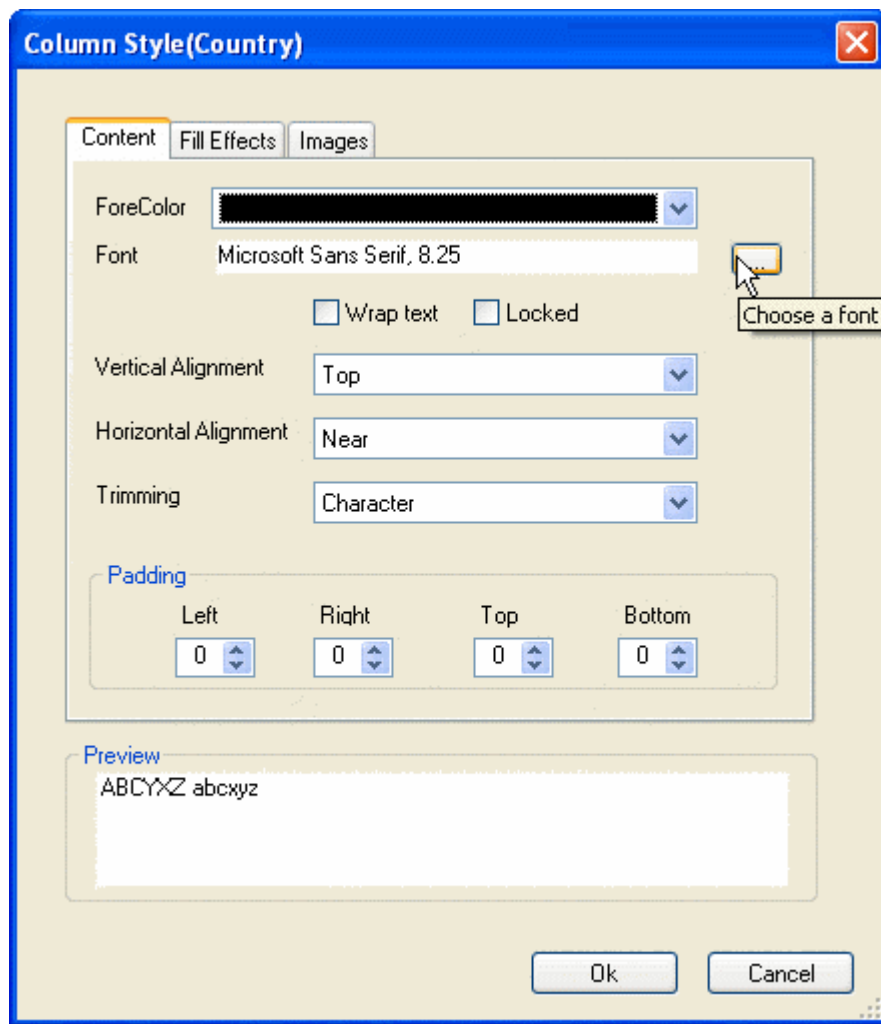
2. Select **Column Style** from the menu.



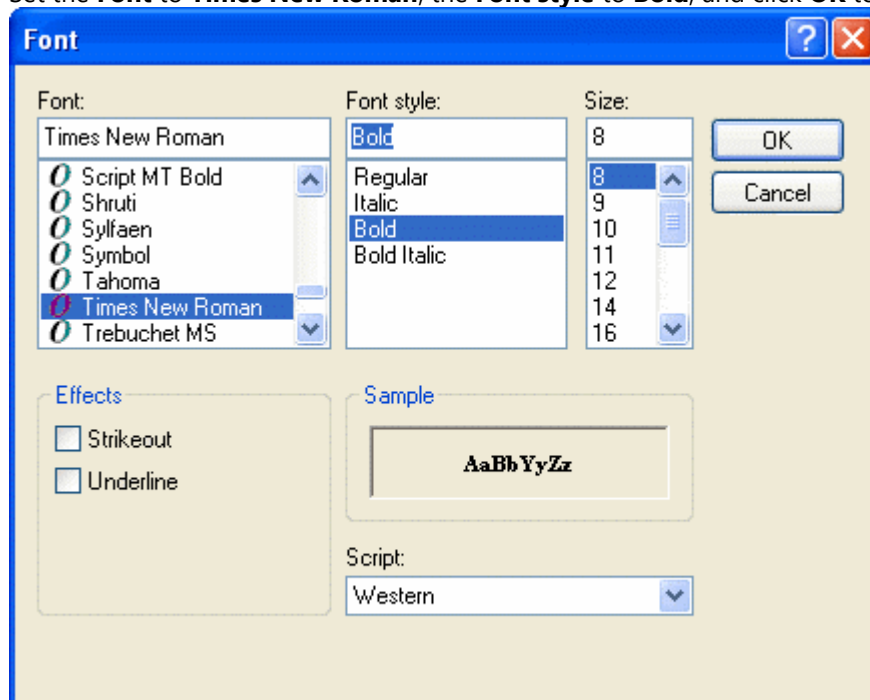
The screenshot shows the 'C1TrueDBGrid Tasks' dialog box with the 'Column Tasks' tab selected. The dialog has a title bar 'C1TrueDBGrid Tasks' and a tab 'Column Tasks'. Below the tab are several settings:

- Select Column:** A dropdown menu with 'Country' selected.
- Column Caption:** A text box containing 'Country'.
- Data Field:** A dropdown menu with 'Country' selected.
- Input Mask:** A text box with an ellipsis button to its right.
- Aggregate:** A dropdown menu with 'None' selected.
- Visible:** A checked checkbox.
- Visible when Grouped:** An unchecked checkbox.
- Edit using DateTimePicker:** A checked checkbox.
- Links:** Four blue links: 'Caption Style...', 'Column Style...' (with a mouse cursor pointing to it), 'Value Items...', and 'C1TrueDBGrid Tasks'.
- Buttons:** Three blue buttons: 'Undock in parent container', 'Add Query...', and 'Preview Data...'.

3. Click the **ellipsis** button after the **Font** property to open the **Font** dialog box.



4. Set the **Font** to **Times New Roman**, the **Font style** to **Bold**, and click **OK** to close the **Font** dialog box.



5. Click **OK** to close the **Column Style(Country)** dialog box.

In the Designer

Alternatively, the font style can also be set through the **C1TrueDBGrid Designer**. To set the font style using the designer:

1. Open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
2. Select the *Country* column by selecting its column header from the right pane.
The column can also be selected by choosing *Country* from the drop-down list on the toolbar.
3. Set the font to **Times New Roman** and click **Bold** on the toolbar.



4. Click **OK** to close the designer.

In Code

To set the font style using code, complete the following steps:

1. Declare a new font variable:

To write code in Visual Basic

Visual Basic

```
Dim fntFont As Font
```

To write code in C#

C#

```
Font fntFont;
```

2. Set the desired column's font to the new font variable:

To write code in Visual Basic

Visual Basic

```
fntFont = New Font("Times New Roman",  
Me.C1TrueDBGrid1.Splits(0).DisplayColumns.Item("Country").Style.Font.Size,  
FontStyle.Bold)  
Me.C1TrueDBGrid1.Splits(0).DisplayColumns.Item("Country").Style.Font = fntFont
```

To write code in C#

C#

```
fntFont = new Font("Times New Roman",  
this.c1TrueDBGrid1.Splits[0].DisplayColumns["Country"].Style.Font.Size,  
FontStyle.Bold);  
this.c1TrueDBGrid1.Splits[0].DisplayColumns["Country"].Style.Font = fntFont;
```

What You've Accomplished

In this example, the rows of the *Country* column are now Times New Roman font and bold:



| First | Last | Country | Birth |
|------------------|-----------|----------------------|------------|
| Isaac | Albeniz | Spain | 5/29/1860 |
| Johann Sebastian | Bach | Germany | |
| Samuel | Barber | United States | 3/9/1910 |
| Bela | Bartok | Hungary | 3/25/1881 |
| Ludwig van | Beethoven | Germany | 12/16/1770 |
| Alban | Berg | Austria | 2/9/1885 |
| Luciano | Berio | Italy | 10/10/1925 |
| Hector | Berlioz | France | 12/11/1803 |
| Leonard | Bernstein | United States | 8/25/1918 |
| Georges | Bizet | France | 10/25/1838 |
| Ernest | Bloch | Switzerland | 7/24/1880 |
| Alexander | Borodin | Russia | 11/12/1833 |
| Johannes | Brahms | Germany | 5/7/1833 |
| Benjamin | Britten | England | 11/22/1913 |

For more information on specifying cell styles, see [Applying Styles to Cells](#).

Aligning the Column Headers

You may choose to align the column headers with or without aligning the column text. In the following example, the caption for the *Last* column has been centered:

To write code in Visual Basic

Visual Basic

```
Me.clTrueDBGrid1.Splits(0).DisplayColumns("Last").HeadingStyle.HorizontalAlignment =
AlignHorzEnum.Center
```

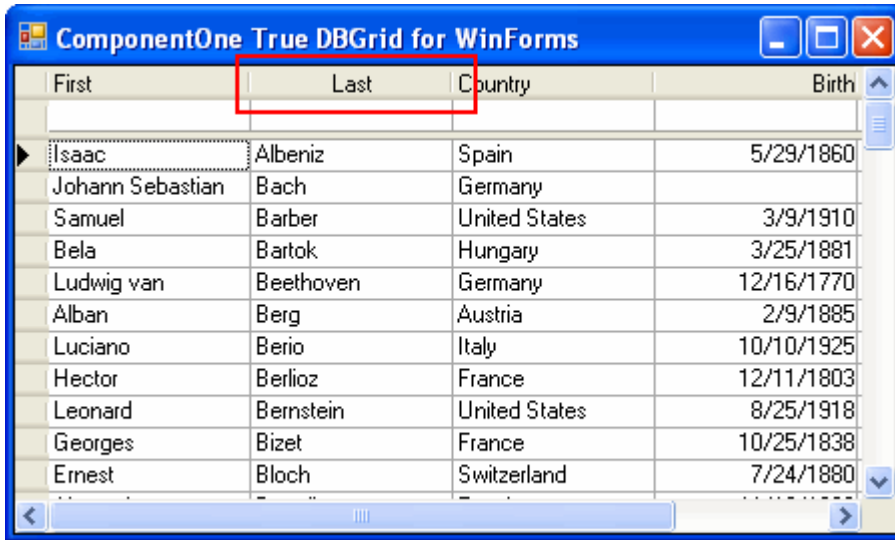
To write code in C#

C#

```
this.clTrueDBGrid1.Splits[0].DisplayColumns["Last"].HeadingStyle.HorizontalAlignment
= AlignHorzEnum.Center;
```

What You've Accomplished

The *First* column is has been center-aligned:



To align all column headers in the grid, loop through each column.

Moving the Focus in Code

At run time the grid cell's focus is usually determined by the user's mouse and keyboard interaction with the grid. However, if you choose to, you can set the column and row of the grid that has focus using the [Col](#) and [Row](#) properties of the grid.

In this topic you'll add two **NumericUpDown** controls to the form. When the value in those boxes changes, the column and row focus of the grid will change.

Complete the following steps:

1. Navigate to the Visual Studio Toolbox and add two **Label** controls and two **NumericUpDown** controls to the form.
2. Arrange **Label1** next to **NumericUpDown1** and **Label2** next to **NumericUpDown2** and set the following properties for the controls:
 - Set **Label1.Text** to "Column:".
 - Set **Label2.Text** to "Row:".
3. Double-click **NumericUpDown1** to create the **ValueChanged** event handler and switch to code view.
4. Add the following code to the **NumericUpDown1_ValueChanged** event:

To write code in Visual Basic

```
Visual Basic
Me.ClTrueDBGrid1.Col = Me.NumericUpDown1.Value
```

To write code in C#

```
C#
this.clTrueDBGrid1.Col = this.numericUpDown1.Value;
```

5. Return to Design view and double-click **NumericUpDown2** to create the **ValueChanged** event handler and switch to code view.
6. Add the following code to the **NumericUpDown2_ValueChanged** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Row = Me.NumericUpDown2.Value
```

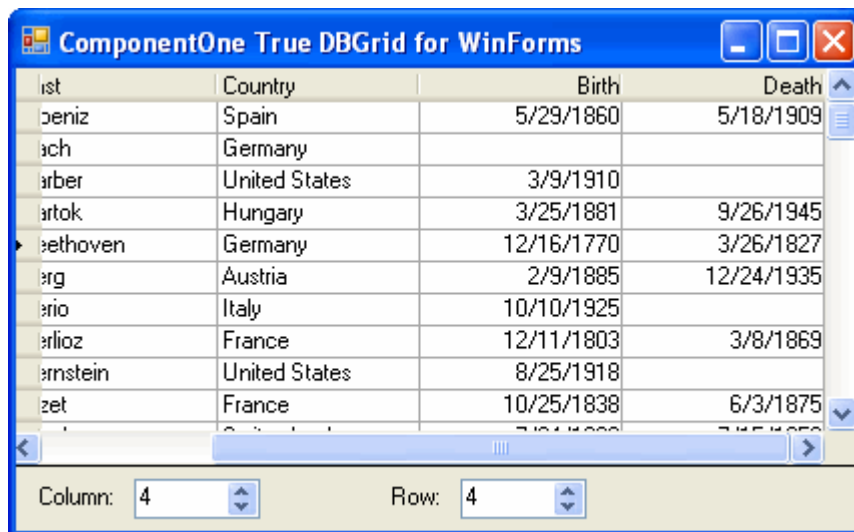
To write code in C#

C#

```
this.c1TrueDBGrid1.Row = this.numericUpDown2.Value;
```

What You've Accomplished

Change the value in the **NumericUpDown** boxes. Note that the focus of the grid changes and the grid scrolls to bring the column and row in focus into view:



Adding Custom Error Checking to C1TrueDBGrid

C1TrueDBGrid displays a message for any errors that occur when building a project. You must switch off the internal error handling.

Complete the following steps:

1. To do this, set the **Handled** property to **True** in the **Error** event of the grid. It will switch off the grid's built-in error checking:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_Error(ByVal sender As Object, ByVal e As  
C1.Win.C1TrueDBGrid.ErrorEventArgs) Handles C1TrueDBGrid1.Error  
    e.Handled = True  
End Sub
```

To write code in C#

C#

```
private void c1TrueDBGrid1_Error(object sender,
C1.Win.C1TrueDBGrid.ErrorEventArgs e)
{
    e.Handled = true;
}
```

2. You can then add your own error-handling code. For example:

To write code in Visual Basic**Visual Basic**

```
Private Sub C1TrueDBGrid1_Error(ByVal sender As Object, ByVal e As
C1.Win.C1TrueDBGrid.ErrorEventArgs) Handles C1TrueDBGrid1.Error
    If C1TrueDBGrid1.Columns(C1TrueDBGrid1.Col).DataField = "CategoryID" Then
        e.Handled = True
        MessageBox.Show("Your User Friendly Message")
    Else
        e.Handled = False
        MessageBox.Show("Enter a string")
    End If
End Sub
```

To write code in C#**C#**

```
private void c1TrueDBGrid1_Error(object sender,
C1.Win.C1TrueDBGrid.ErrorEventArgs e)
{
    if (c1TrueDBGrid1.Columns[c1TrueDBGrid1.Col].DataField ==
"CategoryID")
    {
        e.Handled = true;
        MessageBox.Show("Your User Friendly Message");
    }
    else
    {
        e.Handled = false;
        MessageBox.Show("Enter a string");
    }
}
```

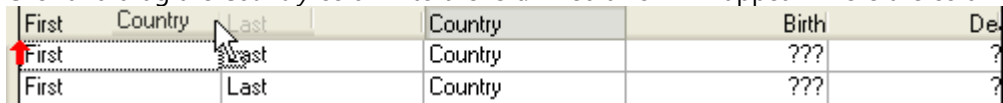
Changing the Column Order in the Grid

To change the column order in the grid, use the **C1TrueDBGrid Designer** or set [RemoveAt](#) and [Insert](#) methods in code.

In the Designer

To change the grid column order in the Designer, complete the following steps:

1. Open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
2. In the designer, select the *Country* column from the right pane.
3. Click and drag the *Country* column to the left. A red arrow will appear where the column can be dropped.



| | | | | | |
|-------|---------|------|---------|-------|----|
| First | Country | Last | Country | Birth | De |
| First | Country | Last | Country | ??? | ? |
| First | Country | Last | Country | ??? | ? |

4. Drop the *Country* column before the *First* column.
5. Click **OK** to close the designer.

In Code

To change the grid column order in code, complete the following steps:

1. Declare the variable for the *Country* column by adding the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Dim dispColumn As C1.Win.C1TrueDBGrid.C1DisplayColumn
dispColumn = Me.C1TrueDBGrid1.Splits(0).DisplayColumns(2)
```

To write code in C#

C#

```
C1.Win.C1TrueDBGrid.C1DisplayColumn dispColumn;
dispColumn = this.c1TrueDBGrid.Splits[0].DisplayColumns[2];
```

2. Move the *Country* column before the *First* column:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Splits(0).DisplayColumns.RemoveAt(2)
Me.C1TrueDBGrid1.Splits(0).DisplayColumns.Insert(0, dispColumn)
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Splits[0].DisplayColumns.RemoveAt(2);
this.c1TrueDBGrid1.Splits[0].DisplayColumns.Insert(0, dispColumn);
```

What You've Accomplished

The *Country* column appears in the grid before the *First* column:



Resizing Columns During Grid Resizing

To expand or shrink columns during grid resizing, set the [SpringMode](#) property to **True** and the [MinWidth](#) property for each column. This can be done either in the designer or in code.

In the Designer

To expand or shrink columns during grid resizing in the designer, complete the following steps:

1. Open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
2. Click the **Split** tab in the left pane.
3. Locate the [SpringMode](#) property and set it to **True**.
Alternatively, the [SpringMode](#) property can also be in the Properties window.
4. Select the *First* column in the right pane by clicking on it.
The column can also be selected by choosing *First* from the drop-down list in the toolbar.
5. Click the **Display Column** tab in the left pane.
6. Locate the [MinWidth](#) property and set it to **50**.
7. Click **OK** to close the designer.

In Code

To expand or shrink columns during grid resizing in code, complete the following steps:

1. Set the [SpringMode](#) property to **True** by adding the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic  
Me.C1TrueDBGrid1.SpringMode = True
```

To write code in C#

C#

```
this.clTrueDBGrid1.SpringMode = true;
```

- Set the [MinWidth](#) property to **50** for the *First* column:

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.Splits(0).DisplayColumns("First").MinWidth = 50
```

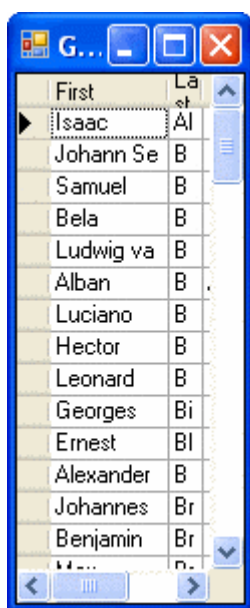
To write code in C#

C#

```
this.clTrueDBGrid1.Splits[0].DisplayColumns["First"].MinWidth = 50;
```

What You've Accomplished

When the grid is resized horizontally, the columns will expand or shrink proportionally, except for the *First* column which will only shrink to 50:




Exporting Grid Data

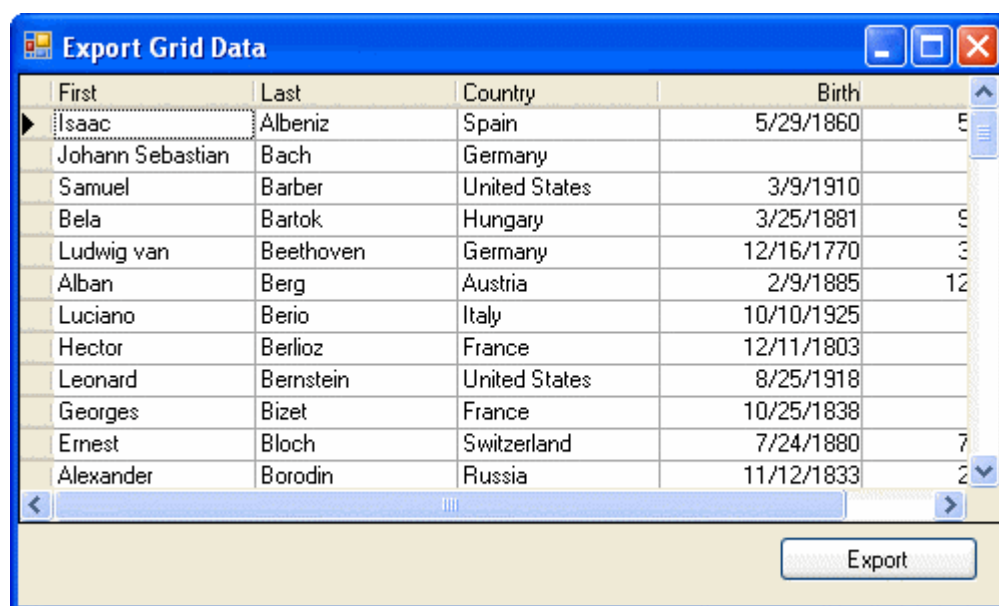
Grid data can be exported as a delimited text, Excel, HTML, PDF, or RTF file. The following table describes the methods used to export each file type:

| File Type | Method | Description |
|----------------|---------------------------------------|---|
| All | ExportTo | Opens a dialog box in which the user can select the export format. |
| Delimited Text | ExportToDelimitedFile | Exports the specified rows from the grid to the specified file as delimited text. |

| | | |
|-------|-------------------------------|------------------------------------|
| Excel | ExportToExcel | Exports the grid to an Excel file. |
| HTML | ExportToHTML | Exports the grid to an HTML file. |
| PDF | ExportToPDF | Exports the grid to a PDF file. |
| RTF | ExportToRTF | Exports the grid to an RTF file. |

 **Note:** C1TrueDBGrid's export feature uses **Reports for WinForms**' components internally, and you may need to reference **Reports for WinForms**' assemblies (C1.Win.C1Report and C1.C1Report) if you are receiving an error related to the assembly.

To set one of the following export methods, add the appropriate code to the **Click** event of the **Export** button:



Exporting To All Available File Types

To set the [ExportTo](#) method, add the following code to the **Click** event of the **Export** button:

To write code in Visual Basic


Visual Basic

```
Me.C1TrueDBGrid1.ExportTo()
```

To write code in C#

C#

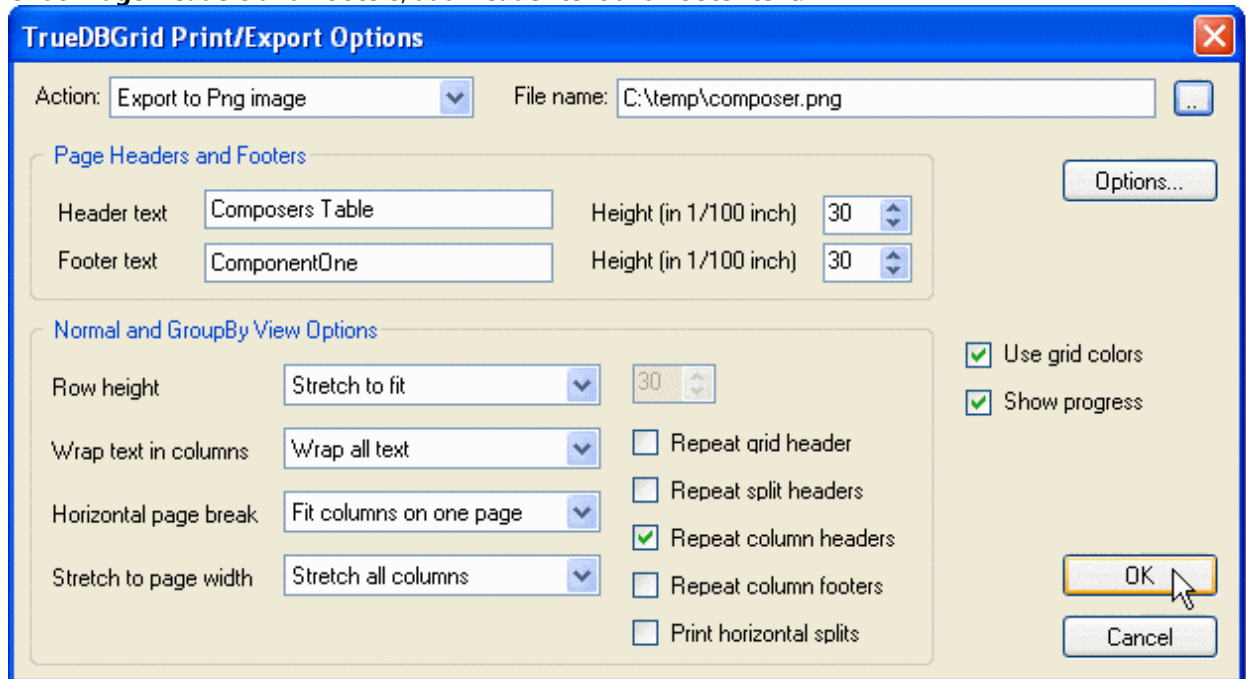
```
this.C1TrueDBGrid1.ExportTo();
```

 **Note:** C1TrueDBGrid's export feature uses **Reports for WinForms**' components internally, and you may need to reference **Reports for WinForms**' assemblies (C1.Win.C1Report and C1.C1Report) if you are receiving an error related to the assembly.

This topic illustrates the following:

Clicking the **Export** button opens the **TrueDBGrid Print/Export Options** dialog box.

1. In the **Action** drop-down list, select the file type, including metafiles and image files.
2. Click the **ellipsis** button next to the **File name** box to open the **Export To** dialog box. Browse to a location to save the file and enter the file name in the **File name** box. Click **OK** to close the **Export To** dialog box.
3. Under **Page Headers and Footers**, add **Header text** and **Footer text**.



4. Click **OK** to export the file.

The final output will look similar to the following image:

Composers Table

| First | Last | Country | Birth | Death |
|------------------|-----------|---------------|------------|------------|
| Isaac | Albeniz | Spain | 5/29/1860 | 5/18/1909 |
| Johann Sebastian | Bach | Germany | | |
| Samuel | Barber | United States | 3/9/1910 | |
| Bela | Bartok | Hungary | 3/25/1881 | 9/26/1945 |
| Ludwig van | Beethoven | Germany | 12/16/1770 | 3/26/1827 |
| Alban | Berg | Austria | 2/9/1885 | 12/24/1935 |
| Luciano | Berio | Italy | 10/10/1925 | |

Exporting to Delimited Text

To set the `ExportToDelimitedFile` method, add the following code to the **Click** event of the **Export** button:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.ExportToDelimitedFile("c:\temp\composers.csv",
```

```
C1.Win.C1TrueDBGrid.RowSelectorEnum.AllRows, ",")
```

To write code in C#

C#

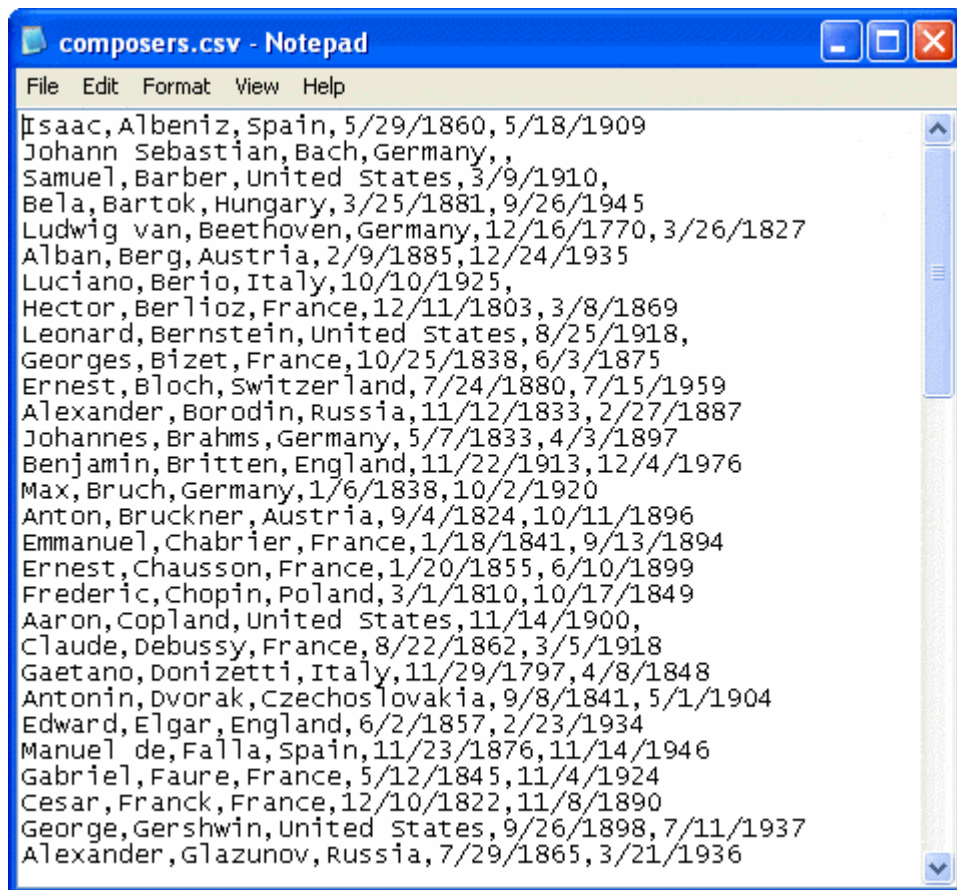
```
this.c1TrueDBGrid1.ExportToDelimitedFile(@"c:\temp\composers.csv",  
C1.Win.C1TrueDBGrid.RowSelectorEnum.AllRows, ",");
```



Note: C1TrueDBGrid's export feature uses **Reports for WinForms**' components internally, and you may need to reference **Reports for WinForms**' assemblies (C1.Win.C1Report and C1.C1Report) if you are receiving an error related to the assembly.

What You've Accomplished

Clicking the **Export** button creates a delimited text file in the temp directory specified in the code above. Each value in the file is separated by a comma:



Exporting to Excel

TrueDBGrid allows exporting grid data to Microsoft Excel format by using either of the following methods:

- **SaveExcel method**
- **ExportToExcel method**

It is important to note that time taken to export grid data to Excel format is very small when using the SaveExcel method. In comparison, the ExportToExcel method takes much longer to export data.

SaveExcel method

To set the [SaveExcel](#) method, add the following code to the Click event of the Export button:

To write code in Visual Basic

```
Visual Basic  
Me.c1TrueDBGrid1.SaveExcel("../GridData.xlsx")
```

To write code in C#

```
C#  
this.c1TrueDBGrid1.SaveExcel("../GridData.xlsx");
```

ExportToExcel method

To set the [ExportToExcel](#) method, add the following code to the Click event of the Export button:

To write code in Visual Basic

```
Visual Basic  
Me.C1TrueDBGrid1.ExportToExcel ("c:\temp\composers.xls")
```

To write code in C#

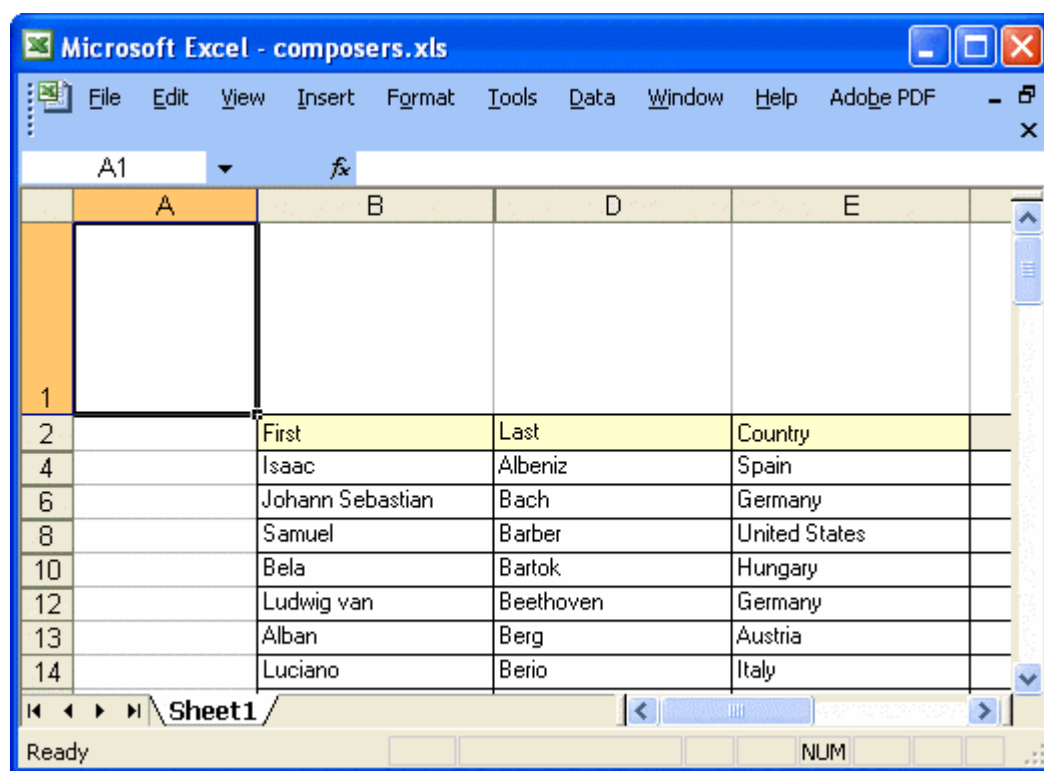
```
C#  
this.c1TrueDBGrid1.ExportToExcel (@ "c:\temp\composers.xls" );
```



Note: C1TrueDBGrid's export feature uses **Reports for WinForms**' components internally, and you may need to reference **Reports for WinForms**' assemblies (C1.Win.C1Report and C1.C1Report) if you are receiving an error related to the assembly.

What You've Accomplished

Clicking the **Export** button creates an Excel file in the temp directory indicated in the code above:



Exporting to HTML

To set the [ExportToHTML](#) method, add the following code to the **Click** event of the **Export** button:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.ExportToHTML("c:\temp\composers.html")
```

To write code in C#

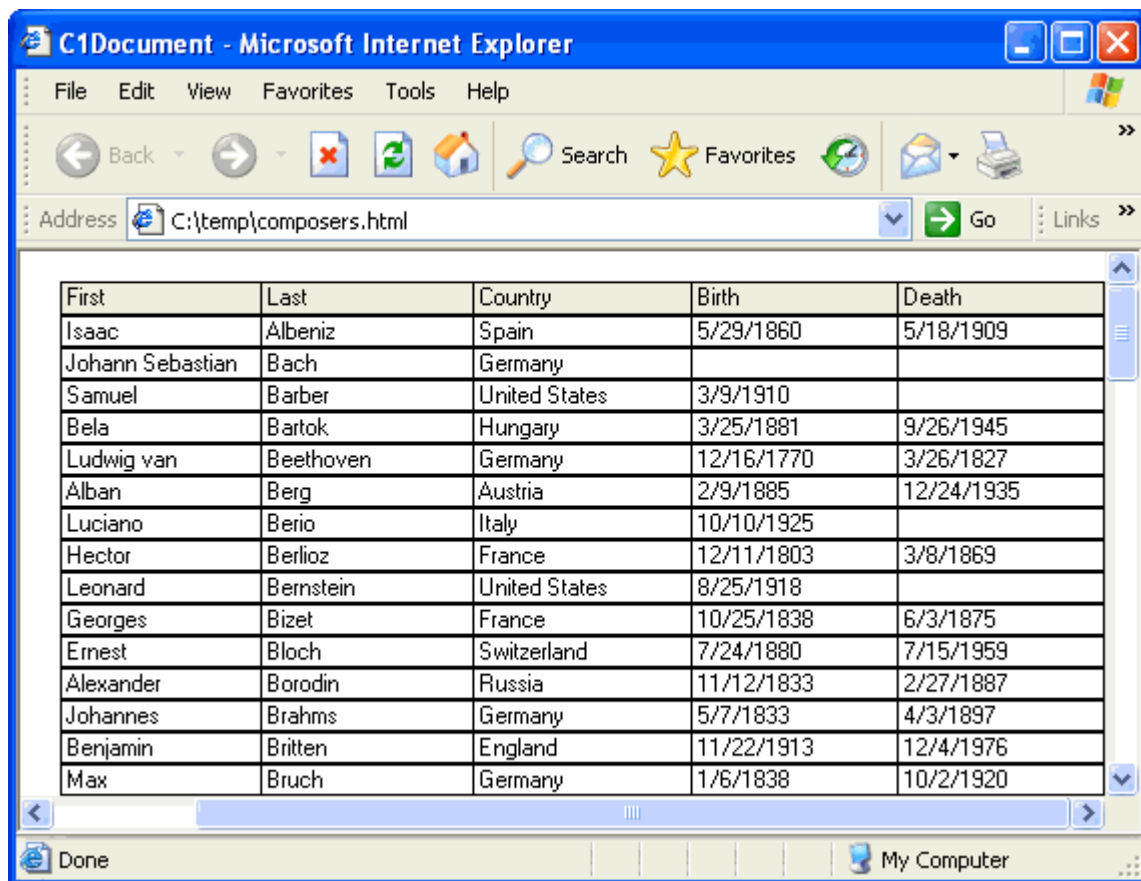
C#

```
this.C1TrueDBGrid1.ExportToHTML(@"c:\temp\composers.html");
```

Note: C1TrueDBGrid's export feature uses **Reports for WinForms**' components internally, and you may need to reference **Reports for WinForms**' assemblies (C1.Win.C1Report and C1.C1Report) if you are receiving an error related to the assembly.

What You've Accomplished

Clicking the **Export** button creates an HTML file in the temp directory indicated in the code above:



Exporting to PDF

To set the [ExportToPDF](#) method, add the following code to the **Click** event of the **Export** button:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.ExportToPDF("c:\temp\composers.pdf")
```

To write code in C#

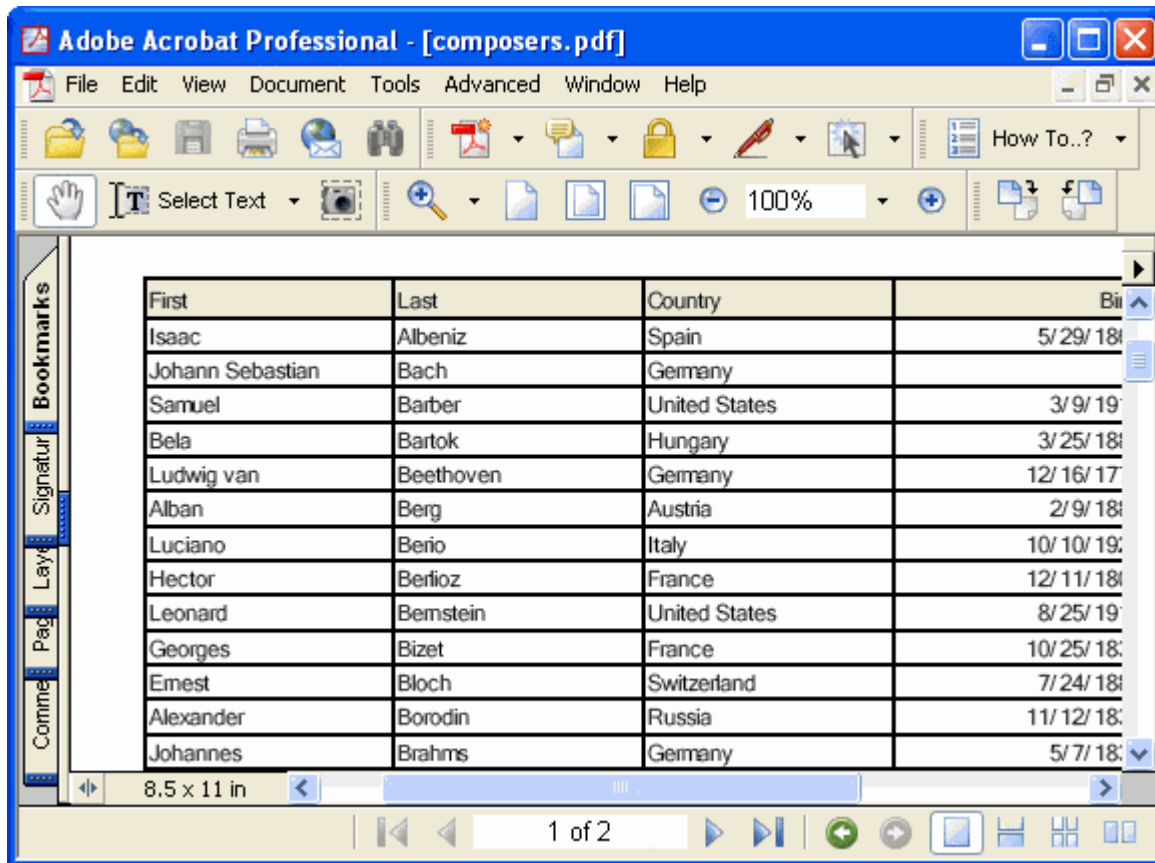
C#

```
this.c1TrueDBGrid1.ExportToPDF(@"c:\temp\composers.pdf");
```

Note: C1TrueDBGrid's export feature uses **Reports for WinForms**' components internally, and you may need to reference **Reports for WinForms**' assemblies (C1.Win.C1Report and C1.C1Report) if you are receiving an error related to the assembly.

What You've Accomplished

Clicking the **Export** button creates a PDF file in the temp directory:



Exporting to RTF

To set the [ExportToRTF](#) method, add the following code to the **Click** event of the **Export** button:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.ExportToRTF("c:\temp\composers.rtf")
```

To write code in C#

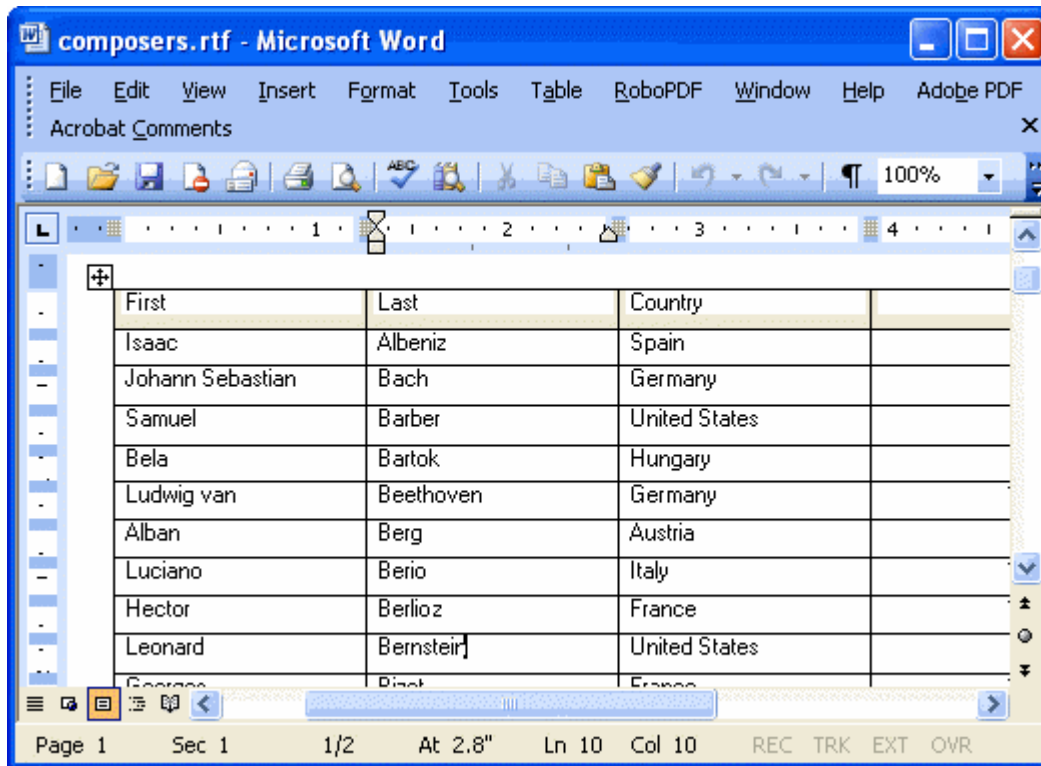
C#

```
this.C1TrueDBGrid1.ExportToRTF(@"c:\temp\composers.rtf");
```

Note: C1TrueDBGrid's export feature uses **Reports for WinForms** components internally, and you may need to reference **Reports for WinForms** assemblies (C1.Win.C1Report and C1.C1Report) if you are receiving an error related to the assembly.

What You've Accomplished

Clicking the **Export** button creates a RTF file in the temp directory indicated in the code above:



Getting the DataRow for a Row Index After Sorting or Filtering

When sorting or filtering is applied to the grid, it uses the underlying **DataView** of the **DataSource** and **DataMember**. To get the **DataRow** for a row index after the sort or filter, access the same underlying list as accessed by the grid with the following code:

To write code in Visual Basic

Visual Basic

```
If Me.C1TrueDBGrid1.FocusedSplit.Rows(Me.C1TrueDBGrid1.Row).RowType =
C1.Win.C1TrueDBGrid.RowTypeEnum.DataRow Then
    Dim dr As System.Data.DataRowView =
    CType(Me.C1TrueDBGrid1(Me.C1TrueDBGrid1.RowBookmark(Me.C1TrueDBGrid1.Row)), System.Data.DataRowView)
End If
```

To write code in C#

C#

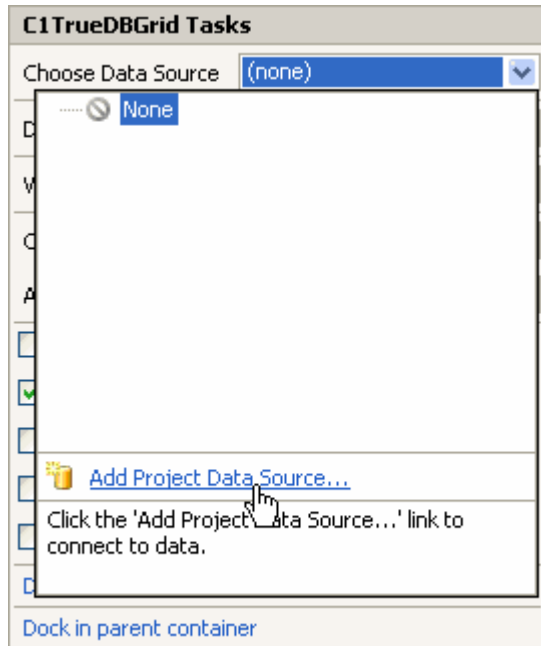
```
if (this.c1TrueDBGrid1.FocusedSplit.Rows[this.c1TrueDBGrid1.Row].RowType ==
C1.Win.C1TrueDBGrid.RowTypeEnum.DataRow)
{
    System.Data.DataRowView dr =
    (System.Data.DataRowView) this.c1TrueDBGrid1[this.c1TrueDBGrid1.RowBookmark(this.c1TrueDBGrid1.Row)];
}
```

Modifying the ConnectionString

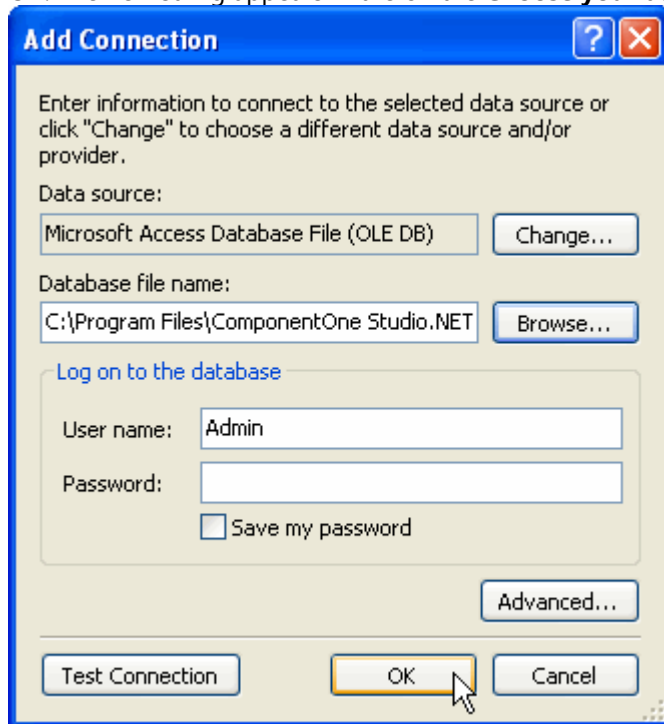
To change the location of the **C1NWind.mdb** reference, you can edit the **ConnectionString** property of the **OleDbConnection**. Note that you can see [Data Binding](#) for more information about binding the grid.

Complete the following steps:

1. In the **C1TrueDBGrid Tasks** menu, select **Add Project Data Source** from the drop-down box next to **Choose Data Source**.

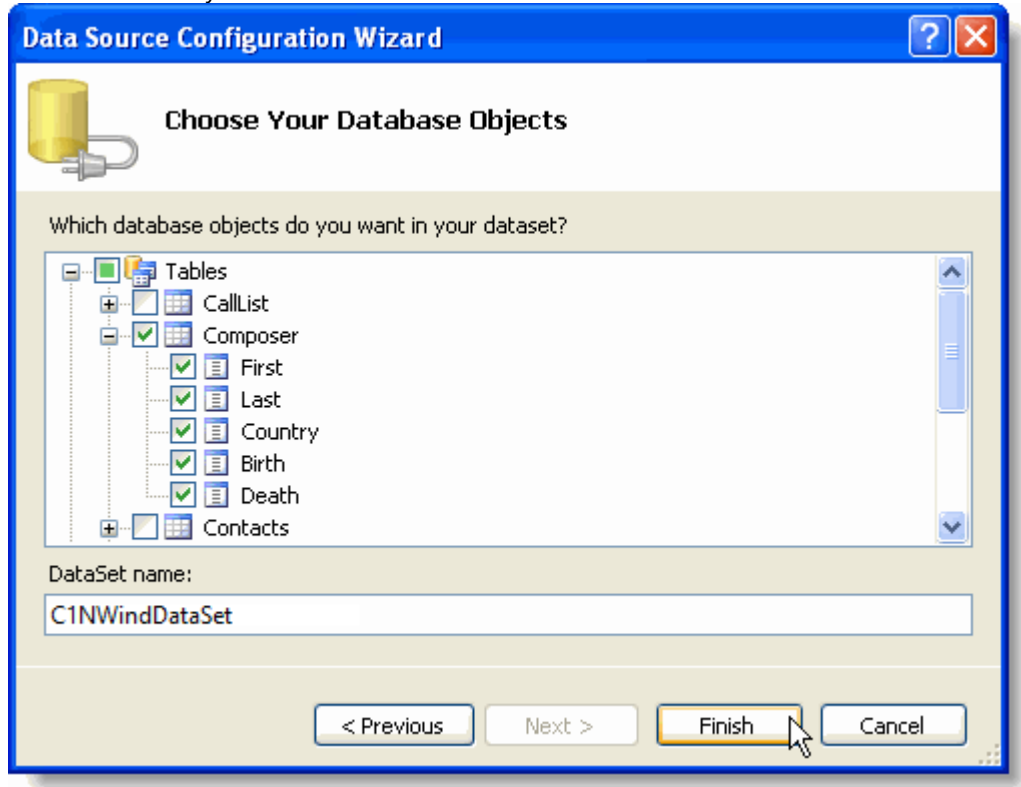


2. The **Data Source Configuration Wizard** appears. Select **Database** on the **Choose a Data Source type** page and click **Next**.
3. Click the **New Connection** button to create a new connection or choose one from the drop-down list.
4. Click the **Browse** button to specify the location of the data and enter the correct login information. Click the **Test Connection** button to make sure that you have successfully connected to the database or server and click **OK**. The new string appears in the on the **Choose your data connection** page.



5. Click the **Next** button to continue. A dialog box will appear asking if you would like to add the data file to your project and modify the connection string. Click **No**.
6. Save the connection string in the application configuration file by checking the **Yes, save the connection as** box and entering a name. Click the **Next** button to continue.

- On the **Choose your database object** page, select the tables and fields that you would like in your dataset. Enter a name for your **DataSet** in the **DataSet name** box and click **Finish** to exit the wizard.



A DataSet and a connection string are added to your project. Additionally, Visual Studio automatically creates the code to fill the DataSet.

Moving to the AddNew Row

To make the AddNew row the active row when the program runs, use the [AllowAddNew](#) property, and the [MoveLast](#) and **Select** methods.

Complete the following steps:

- Set the [AllowAddNew](#) property to **True** either in the Properties window or by adding the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.AllowAddNew = True
```

To write code in C#

C#

```
this.c1TrueDBGrid1.AllowAddNew = true;
```

- Move to the last record in the grid by adding following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.MoveLast()
```

To write code in C#

C#

```
this.ClTrueDBGrid1.MoveLast();
```

3. Move to the AddNew row by adding following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.Row = Me.ClTrueDBGrid1.Row + 1
```

To write code in C#

C#

```
this.ClTrueDBGrid1.Row = this.ClTrueDBGrid1.Row + 1;
```

4. Set focus to the grid by adding following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.Select()
```

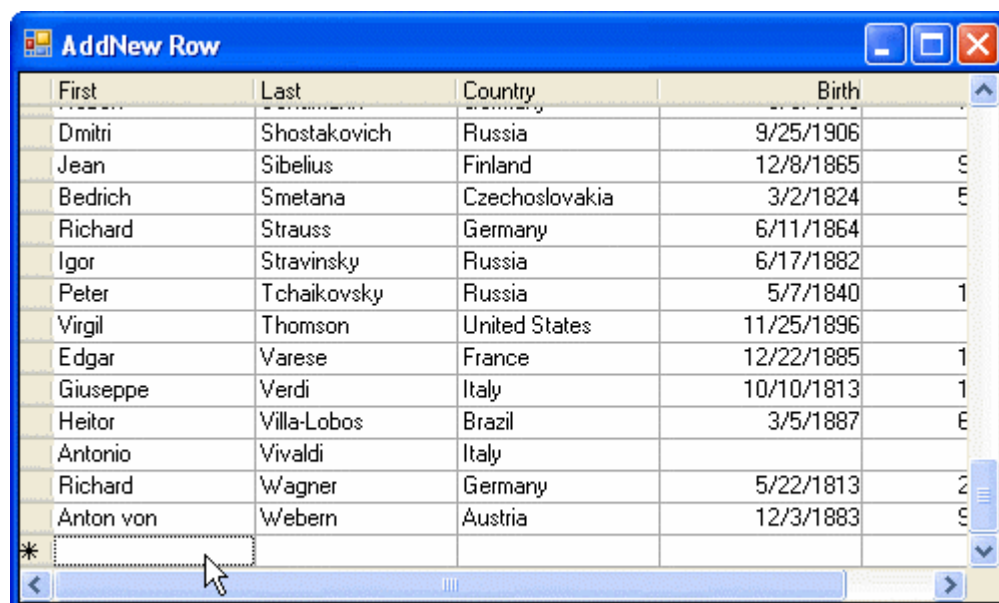
To write code in C#

C#

```
this.ClTrueDBGrid1.Select();
```

What You've Accomplished

When the program runs, the active row is the AddNew row:



Saving the Layout of the Grid

To save the layout of the grid, use the [SaveLayout](#) method, which will save the layout in an XML file. This can be done either in the designer or in code.

In the Designer

Complete the following steps to save the layout of the grid:

1. Open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
2. In the designer, click **Save Layout** on the toolbar to open the **Save As** dialog box.



3. Browse to a location and enter a file name in the **File Name** box.
4. Click **Save** to save the layout as an XML file.
5. Click **OK** to close the designer.

In Code

Add the following code to the **Click** event of a button to save the layout of the grid:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.SaveLayout("c:\temp\ComposerLayout.xml")
```

To write code in C#

C#

```
this.c1TrueDBGrid1.SaveLayout(@"c:\temp\ComposerLayout.xml");
```

What You've Accomplished

You've learned how to use the [SaveLayout](#) method to save the layout in an XML file.

Searching for Entries in a Column

To search for entries in a column using an incremental search, add a **Timer** component to the form, then set the **KeyPress** and **Tick** events.

Complete the following steps:

1. Add a **Timer** component from the Visual Studio Toolbox to the form.



2. Set the Timer's **Interval** property to 1 second.

In the Designer

Locate the **Interval** property for Timer1 in the Properties window and set it to **1000**.

In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic
Me.Timer1.Interval = 1000
```

To write code in C#

```
C#
this.timer1.Interval = 1000;
```

3. Declare the search string variable at the form level:

To write code in Visual Basic

```
Visual Basic
Dim searchString As String = String.Empty
```

To write code in C#

```
C#
string searchString = string.Empty;
```

4. Add the **KeyPress** event:

To write code in Visual Basic

```
Visual Basic
Private Sub C1TrueDBGrid1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles C1TrueDBGrid1.KeyPress
```



```

' Handle the keystroke.
e.Handled = True

Me.searchString += e.KeyChar
Dim count As Integer = Me.C1TrueDBGrid1.Splits(0).Rows.Count
Dim start As Integer = Me.C1TrueDBGrid1.Row
Dim current As Integer = (start + 1) Mod count

' Stop if search returns to the starting position.
While current <> start

    ' Get the value.
    Dim s As String = Me.C1TrueDBGrid1(current,
Me.C1TrueDBGrid1.Col).ToString()

    ' If a match is found, exit.
    If s.Substring(0, Me.searchString.Length).ToUpper() =
Me.searchString.ToUpper() Then
        Exit While
    End If

    ' Search the next row, wrapping the column if needed.
    current = (current + 1) Mod count
End While

' Update the grid's current row.
Me.C1TrueDBGrid1.Row = current

' Highlight the entry.
Me.C1TrueDBGrid1.MarqueeStyle =
C1.Win.C1TrueDBGrid.MarqueeEnum.HighlightCell

' Clear the search string at 1 second.
Me.Timer1.Enabled = True
End Sub

```

To write code in C#

```

C#
private void c1TrueDBGrid1_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
{
    // Handle the keystroke.
    e.Handled = true;

    this.searchString += e.KeyChar;
    int count = this.c1TrueDBGrid1.Splits[0].Rows.Count;
    int start = this.c1TrueDBGrid1.Row;
    int current = (start + 1) % count;

    // Stop if search returns to the starting position.

```

```
while( current != start )
{
    // Get the value.
    string s = this.clTrueDBGrid1[current,
this.clTrueDBGrid1.Col].ToString();

    // If a match is found, exit.
    if( s.Substring(0, this.searchString.Length).ToUpper() ==
this.searchString.ToUpper() )
        break;

    // Search the next row, wrapping the column if needed.
    current = (current + 1) % count;
}

// Update the grid's current row.
this.clTrueDBGrid1.Row = current;

// Highlight the entry.
this.clTrueDBGrid1.MarqueeStyle =
Cl.Win.ClTrueDBGrid.MarqueeEnum.HighlightCell;

// Clear the search string at 1 second.
this.timer1.Enabled = true;
}
```

5. Add the **Tick** event for the timer:

To write code in Visual Basic

Visual Basic

```
Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Timer1.Tick
    Me.searchString = String.Empty
    Me.Timer1.Enabled = False
End Sub
```

To write code in C#

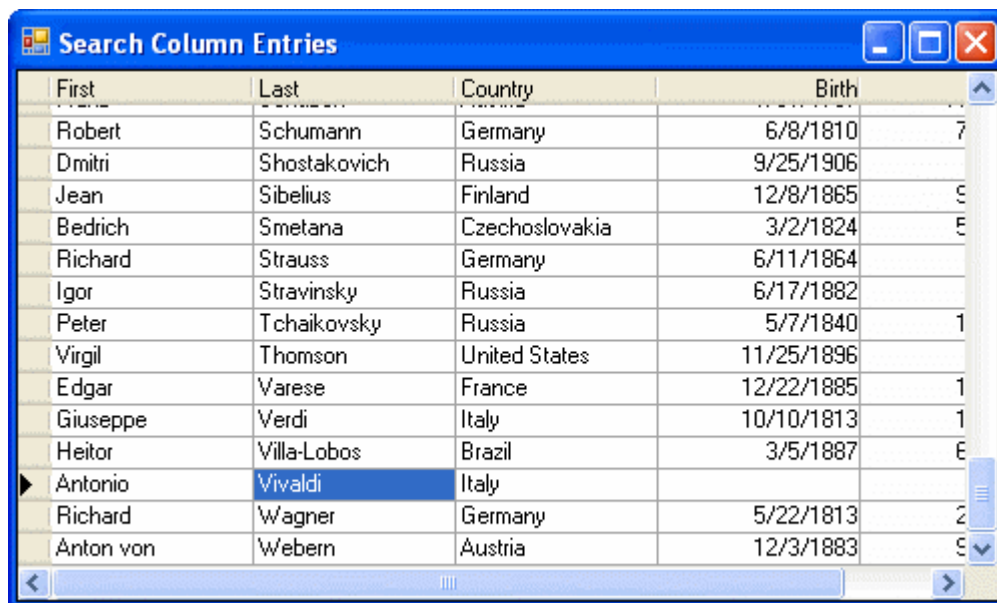
C#


```
private void timer1_Tick(object sender, System.EventArgs e)
{
    this.searchString = string.Empty;
    this.timer1.Enabled = false;
}
```

What You've Accomplished

As the user types, the search will highlight the cell containing that letter. In this example, typing **V** in the *Last* column highlights "Varese".

If more than one entry begins with the same letter, typing the next letter will highlight the entry with those letters. For example, typing **Viv** in the *Last* column will highlight "Vivaldi":



 **Note:** After 1 second, the search string will reset.

Setting Default Values for New Rows

To set default values for new rows, set the column's [Value](#) property in the [OnAddNew](#) event. This is useful if adding multiple rows with similar information.

Complete the following steps:

1. Set the [AllowAddNew](#) property to **True**.

In the Designer

Locate the [AllowAddNew](#) property in the Properties window and set it to **True**.

In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic
Me.C1TrueDBGrid1.AllowAddNew = True
```

To write code in C#

```
C#
this.c1TrueDBGrid1.AllowAddNew = true;
```

2. Add the following [OnAddNew](#) event to the form:

To write code in Visual Basic

Visual Basic

```
Private Sub ClTrueDBGrid1_OnAddNew(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ClTrueDBGrid1.OnAddNew
    Me.ClTrueDBGrid1.Columns("Country").Value = "United States"
End Sub
```

To write code in C#

C#

```
private void clTrueDBGrid1_OnAddNew(object sender, System.EventArgs e)
{
    this.clTrueDBGrid1.Columns["Country"].Value = "United States";
}
```

What You've Accomplished

The value in the *Country* column automatically adds "United States" when a new row is added:

| First | Last | Country | Birth | |
|-------------|-------------|---------------|------------|---|
| Peter | Tchaikovsky | Russia | 5/7/1840 | 1 |
| Virgil | Thomson | United States | 11/25/1896 | |
| Edgar | Varese | France | 12/22/1885 | 1 |
| Giuseppe | Verdi | Italy | 10/10/1813 | 1 |
| Heitor | Villa-Lobos | Brazil | 3/5/1887 | 6 |
| Antonio | Vivaldi | Italy | | |
| Richard | Wagner | Germany | 5/22/1813 | 2 |
| Anton von | Webern | Austria | 12/3/1883 | 9 |
| Duke | Ellington | United States | | |
| Stephen | Foster | United States | | |
| Scott | Joplin | United States | | |
| John Philip | Sousa | United States | | |
| | | United States | | |
| * | | | | |

Displaying a Column Total in the Footer

You can easily display a sum of all values in a column in the footer of a grid. To do so, you would need to make the column footers visible by setting the [ColumnFooters](#) property to **True**; you would then create a function to calculate the sum of the column. Note that in the following example, the grid has been bound to the *Products* table in the Northwind database.

Complete the following steps to calculate the total of the *UnitsInStock* column:

1. Add the following code in the Code Editor:

To write code in Visual Basic

Visual Basic

```
Public Sub CalculateFooter()  
    Dim i As Integer  
    Dim sum As Double  
    For i = 0 To Me.ClTrueDBGrid1.Splits(0).Rows.Count - 1  
        sum += Me.ClTrueDBGrid1.Columns("UnitsInStock").CellValue(i)  
    Next  
    Me.ClTrueDBGrid1.Columns("UnitsInStock").FooterText = sum  
End Sub
```

To write code in C#

```
C#  
  
public void CalculateFooter()  
{  
    int i = 0;  
    double sum = 0;  
    for (i = 0; i <= this.clTrueDBGrid1.Splits[0].Rows.Count - 1; i++)  
    {  
        sum +=  
Convert.ToDouble(this.clTrueDBGrid1.Columns["UnitsInStock"].CellValue(i));  
    }  
    this.clTrueDBGrid1.Columns["UnitsInStock"].FooterText =  
Convert.ToString(sum);  
}
```

This code creates the **CalculateFooter** function to calculate the total of the *UnitsInStock* column.

2. Add the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic  
  
Me.ClTrueDBGrid1.ColumnFooters = True  
CalculateFooter()
```

To write code in C#

```
C#  
  
clTrueDBGrid1.ColumnFooters = true;  
CalculateFooter();
```

This code sets the visibility of the column footer and initializes the **CalculateFooter** function.

What You've Accomplished

The column total for the *UnitsInStock* column is now displayed in the grid's footer:

| ProductID | ProductName | QuantityPerUnit | UnitPrice | UnitsInStock | UnitsOnOrder |
|-----------|----------------|---------------------|-----------|--------------|--------------|
| 14 | Tofu | 40 - 100 g pkgs. | 23.25 | 35 | |
| 15 | Genen Shouyu | 24 - 250 ml bottles | 15.5 | 39 | |
| 16 | Pavlova | 32 - 500 g boxes | 17.45 | 29 | |
| 17 | Alice Mutton | 20 - 1 kg tins | 39 | 0 | |
| 18 | Carnarvon Tige | 16 kg pkg. | 62.5 | 42 | |
| 19 | Teatime Chocol | 10 boxes x 12 piec | 9.2 | 25 | |
| 20 | Sir Rodney's M | 30 gift boxes | 81 | 40 | |
| 21 | Sir Rodney's S | 24 pkgs. x 4 piece | 10 | 3 | |
| 22 | Gustaf's Knäck | 24 - 500 g pkgs. | 21 | 104 | |
| 23 | Tunnbröd | 12 - 250 g pkgs. | 9 | 61 | |
| 24 | Guaraná Fantá | 12 - 355 ml cans | 4.5 | 20 | |
| 25 | NuNuCa Nuß- | 20 - 450 g glasses | 14 | 76 | 3119 |

Displaying the Current Column and Row

Using the [Row](#) and [Col](#) properties you can get the index of the currently selected cell's row and column. In the following example, you'll add two text boxes to your grid application, one that displays the currently selected row and another displaying the current column.

Complete the following steps to display the current row and column:

1. From the Visual Studio Toolbox add two **Label** and two **TextBox** controls.
2. Resize and arrange the controls so that **Label1** is next to **TextBox1** and **Label2** is next to **TextBox2**.
3. In the Properties window, set the following properties:
 - Set **Label1**'s **Text** property to "Row".
 - Set **Label2**'s **Text** property to "Column".
4. Add the following **RowColChange** event in the Code Editor:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TrueDBGrid1_RowColChange(ByVal sender As System.Object, ByVal e As
C1.Win.C1TrueDBGrid.RowColChangeEventArgs) Handles C1TrueDBGrid1.RowColChange
    Me.TextBox1.Text = C1TrueDBGrid1.Row
    Me.TextBox2.Text = C1TrueDBGrid1.Col
End Sub
```

To write code in C#

C#

```
private void c1TrueDBGrid1_RowColChange(object sender, RowColChangeEventArgs e)
{
    this.textBox1.Text = c1TrueDBGrid1.Row;
    this.textBox2.Text = c1TrueDBGrid1.Col;
}
```

This code will set the current row and column indexes to appear in the text boxes.

What You've Accomplished

Run your application and observe that the row and column text boxes display the row and column index for the selected grid cell:



Choose a different cell and note that the text in the text boxes changes to display the currently selected cell's row and column index.

Displaying the Date and Time in a Column

In previous versions of **True DBGrid for WinForms**, the default behavior in a column with a **DataType** of **DateTime** was to display both the date and the time in the column. Currently the default behavior is to display only the date. In the following steps, you'll set the column's **NumberFormat** property to "g" (which displays the short date and short time according to your current culture's format) for both the data and time to be displayed and you'll disable the **DateTimePicker** used to edit the date and time at run time.

In the Designer

Complete the following steps to display both the date and the time in the column:

1. Click the **ellipsis** button next to the **Columns** collection in the Properties window to open the **C1TrueDBGrid Designer**. For information on how to access the **C1TrueDBGrid Designer**, see [Accessing the C1TrueDBGrid Designer](#).
2. In the designer's right pane, select the column you wish to change.
3. In the left pane, select the **Column** tab to view the column's properties.
4. In the properties grid, select the drop-down arrow next to the column's **NumberFormat** property and set it to "g".
5. Select the drop-down arrow next to the column's **EnableDateTimeEditor** property and set it to **False**.
6. Click **OK** to save your changes and close the designer.

In Code

Add the following code to the **Form_Load** event to display both the date and the time in the second column:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Columns(1).EnableDateTimeEditor = False  
Me.C1TrueDBGrid1.Columns(1).NumberFormat = "g"
```

To write code in C#

C#

```
this.c1TrueDBGrid1.Columns[1].EnableDateTimeEditor = false;  
this.c1TrueDBGrid1.Columns[1].NumberFormat = "g";
```

What You've Accomplished

The selected column displays both the date and the time.

Programmatically Entering Edit Mode

At run time cell edit mode is usually entered by the user's mouse and keyboard interaction with the grid. However, if you choose, you can set the currently focused cell to enter edit mode in code. To enter edit mode, simply set the [EditActive](#) property to **True**.

In the following steps you'll add two labels and text boxes to your project to choose a cell to edit, a button to change focus to that cell, and another button that enters the focused cell into edit mode.

Complete the following steps:

1. Navigate to the Visual Studio Toolbox and add two **Label** controls and two **TextBox** controls to the form.
2. Arrange **Label1** next to **TextBox1** and **Label2** next to **TextBox2** and, in the Properties window, set the following properties for the controls:
 - Set **Label1.Text** to "Column:".
 - Set **TextBox1.Text** to "0".
 - Set **Label2.Text** to "Row:".
 - Set **TextBox2.Text** to "0".
3. Navigate to the Visual Studio Toolbox and add two **Button** controls to the form.
4. Arrange the **Button** controls next to the **Label** and **TextBox** controls, and set the following properties in the Properties window:
 - Set **Button1.Text** to "Set Focus".
 - Set **Button2.Text** to "Edit Cell".
5. Double-click **Button1** to create the **Click** event handler and switch to code view.
6. Add the following code to the **Button1_Click** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.Col = Me.TextBox1.Text  
Me.C1TrueDBGrid1.Row = Me.TextBox2.Text
```

To write code in C#

C#

```
this.clTrueDBGrid1.Col = this.textBox1.Text;
this.clTrueDBGrid1.Row = this.textBox2.Text;
```

- Return to Design view and double-click **Button2** to create the **Click** event handler and switch to code view.
- Add the following code to the **Button2_Click** event:

To write code in Visual Basic

Visual Basic

```
Me.ClTrueDBGrid1.EditActive = True
```

To write code in C#

C#

```
this.clTrueDBGrid1.EditActive = true;
```

What You've Accomplished

Using the textboxes and buttons, you can change the cell that is in focus, and you can enter edit mode on the selected cell. Complete the following:

- Run your application.
- Change the values in the **Column** and **Row** text boxes, for example to "2" and "3", and click the **Set Focus** button.
The focus of the grid changes and, if needed, the grid scrolls to bring the column and row in focus into view.
- Click the **Edit Cell** button.

The selected cell will enter edit mode:



Changing the Filter Language

To change the language used in the column filter editor, you can use the [Language](#) property.

1. Right-click your grid and select **Properties** to view the Visual Studio Properties window.
2. Confirm that the [AllowFilter](#) property is set to **True**.
3. Click the drop-down arrow next to the Language property and select a language (for example, **Danish**).
4. Run the project and click the drop-down arrow on one of the column headers to open the column filter editor. The language of the column filter editor matches the language specified in the Language property.

In Code

Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.C1TrueDBGrid1.AllowFilter = True  
Me.C1TrueDBGrid1.Language = C1.Util.Localization.Language.Danish
```

To write code in C#

C#

```
this.c1TrueDBGrid1.AllowFilter = true;  
this.c1TrueDBGrid1.Language = C1.Util.Localization.Language.Danish;
```

This topic illustrates the following:

Notice the language of the column filter editor matches the language specified in the [Language](#) property.

Creating a Custom Print Preview

You can create a custom print preview and customize how your grid will appear when printed. You can do this using the [Init](#) method. To override properties like **FormBorderStyle**, **MaximizeBox**, **MinimizeBox**, **ControlBox** and so on of a Form inherited from `C1.Win.C1TrueDBGrid.PrintForm`, override the **Init** method of the **PrintForm**. First call the `base.Init()`, then set the properties you want.

Complete the following steps:

1. Navigate to the Toolbox and double-click the **SplitContainer** panel to add it to the Form.
2. Navigate to the Properties window and set the **SplitContainer** panel's **Orientation** property to **Horizontal**.
3. Click in the top panel of the **SplitContainer**, navigate to the Toolbox and double-click the **Button** control to add it to the application.
4. In the Properties window, set the **Button** control's **Text** property to "Preview".
5. Click in the bottom panel of the `C1SplitContainer`, navigate to the Toolbox, and locate and then double-click the **C1TrueDBGrid** control to add it to the application.
6. Click the **C1TrueDBGrid** control's smart tag and choose the **Dock in Parent Container** option from the **Tasks** menu.
7. Right-click the project in the Solution Explorer and select **Add Reference**. In the **Add Reference** dialog box, locate and select the **C1.C1Report** and **C1.Win.C1Report** assemblies and click **OK**. This is required for the print preview.

8. Double-click the **Form** to switch to Code view and create the **Form_Load** event handler.
9. Add the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic  
FillGrid()
```

To write code in C#

```
C#  
FillGrid();
```

10. Add the **FillGrid** event just below the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic  
Private Sub FillGrid()  
    Dim maxrows As Integer = 5  
  
    Dim dt As New DataTable("testdatatable")  
  
    Dim dc As DataColumn  
    Dim dr As DataRow  
  
    ' set up an integer column  
    dc = New DataColumn()  
    dc.DataType = System.Type.GetType("System.DateTime")  
    dc.ColumnName = "DT1"  
    dt.Columns.Add(dc)  
  
    ' do string  
    dc = New DataColumn()  
    dc.DataType = System.Type.GetType("System.DateTime")  
    dc.ColumnName = "DT2"  
    dt.Columns.Add(dc)  
  
    ' do string  
    dc = New DataColumn()  
    dc.DataType = System.Type.GetType("System.DateTime")  
    dc.ColumnName = "DT3"  
    dt.Columns.Add(dc)  
  
    Dim rnd As New Random()  
    For i As Integer = 0 To maxrows - 1  
        dr = dt.NewRow()  
        dr("DT1") = DateTime.Now.AddDays(i)  
        dr("DT2") = DateTime.Now.AddMonths(i)  
        dr("DT3") = DateTime.Now.AddYears(i)  
        dt.Rows.Add(dr)  
    Next
```

```
Me.ClTrueDBGrid1.DataSource = dt
Me.ClTrueDBGrid1.Columns("DT1").EnableDateTimeEditor = True
Me.ClTrueDBGrid1.Columns("DT2").EnableDateTimeEditor = True
Me.ClTrueDBGrid1.Columns("DT3").EnableDateTimeEditor = True
End Sub
```

To write code in C#

C#

```
private void FillGrid()
{
    int maxrows = 5;

    DataTable dt = new DataTable("testdatatable");

    DataColumn dc;
    DataRow dr;

    // set up an integer column
    dc = new DataColumn();
    dc.DataType = System.Type.GetType("System.DateTime");
    dc.ColumnName = "DT1";
    dt.Columns.Add(dc);

    // do string
    dc = new DataColumn();
    dc.DataType = System.Type.GetType("System.DateTime");
    dc.ColumnName = "DT2";
    dt.Columns.Add(dc);

    // do string
    dc = new DataColumn();
    dc.DataType = System.Type.GetType("System.DateTime");
    dc.ColumnName = "DT3";
    dt.Columns.Add(dc);

    Random rnd = new Random();
    for (int i = 0; i < maxrows; i++)
    {
        dr = dt.NewRow();
        dr["DT1"] = DateTime.Now.AddDays(i);
        dr["DT2"] = DateTime.Now.AddMonths(i);
        dr["DT3"] = DateTime.Now.AddYears(i);
        dt.Rows.Add(dr);
    }
    this.ClTrueDBGrid1.DataSource = dt;
    this.ClTrueDBGrid1.Columns["DT1"].EnableDateTimeEditor = true;
    this.ClTrueDBGrid1.Columns["DT2"].EnableDateTimeEditor = true;
    this.ClTrueDBGrid1.Columns["DT3"].EnableDateTimeEditor = true;
}
```

11. In the Solution Explorer, right-click the project and select **Add | Windows Form**. In the **Add New Item** dialog

- box, name the form "PrintForm1" and click the **Add** button.
- Double-click the new form to switch to Code view.
 - Edit the initial class declaration to inherit from C1.Win.C1TrueDBGrid.PrintForm:

To write code in Visual Basic

Visual Basic

```
Public Class PrintForm1
    Inherits C1.Win.C1TrueDBGrid.PrintForm
```

To write code in C#

C#

```
public partial class PrintForm1 : C1.Win.C1TrueDBGrid.PrintForm
```

- Add the following code below the class declaration:

To write code in Visual Basic

Visual Basic

```
Protected Overrides Sub Init()
    MyBase.Init()
    FormBorderStyle = FormBorderStyle.Sizable
    Me.ControlBox = True
    Me.MinimizeBox = False
    Me.MaximizeBox = False
End Sub
```

To write code in C#

C#

```
protected override void Init()
{
    base.Init();
    FormBorderStyle = FormBorderStyle.Sizable;
    this.ControlBox = true;
    this.MinimizeBox = false;
    this.MaximizeBox = false;
}
```

- Return to **Form1** in Design view and double-click the **Button** to switch to Code view and create the **Button_Click** event handler.
- Add the following code to the **Button_Click** event handler, making sure to replace "ProjectName" with the name of your project:

To write code in Visual Basic

Visual Basic

```
C1TrueDBGrid1.PrintInfo.PreviewFormClassName = "ProjectName.PrintForm"
C1TrueDBGrid1.PrintInfo.PrintPreview()
```

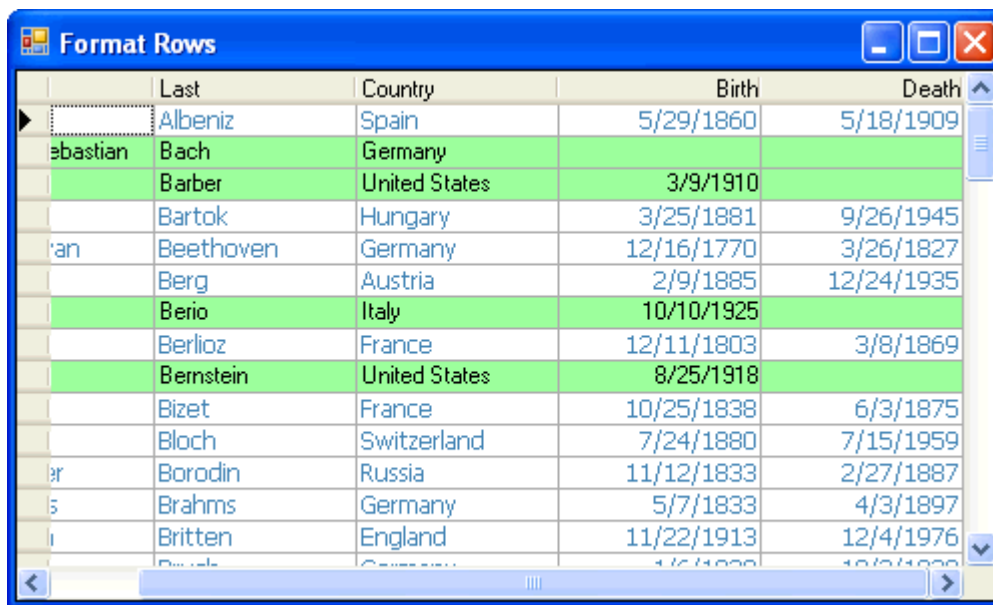
To write code in C#

```
C#
```

```
clTrueDBGrid1.PrintInfo.PreviewFormClassName = "ProjectName.PrintForm1";  
clTrueDBGrid1.PrintInfo.PrintPreview();
```

What You've Accomplished

Run the application and notice the application appears with a button and grid displaying data. Click the Preview button and observe that a customized print preview form appears. The form only includes the **Close** button and not the **Minimize** and **Maximize** buttons.



The screenshot shows a Windows application window titled "Format Rows". It contains a table with five columns: "Last", "Country", "Birth", and "Death". The table lists various composers, with some rows highlighted in green. The window has a standard Windows title bar with minimize, maximize, and close buttons.

| | Last | Country | Birth | Death |
|-----------|-----------|---------------|------------|------------|
| | Albeniz | Spain | 5/29/1860 | 5/18/1909 |
| Sebastian | Bach | Germany | | |
| | Barber | United States | 3/9/1910 | |
| | Bartok | Hungary | 3/25/1881 | 9/26/1945 |
| an | Beethoven | Germany | 12/16/1770 | 3/26/1827 |
| | Berg | Austria | 2/9/1885 | 12/24/1935 |
| | Berio | Italy | 10/10/1925 | |
| | Berlioz | France | 12/11/1803 | 3/8/1869 |
| | Bernstein | United States | 8/25/1918 | |
| | Bizet | France | 10/25/1838 | 6/3/1875 |
| | Bloch | Switzerland | 7/24/1880 | 7/15/1959 |
| er | Borodin | Russia | 11/12/1833 | 2/27/1887 |
| s | Brahms | Germany | 5/7/1833 | 4/3/1897 |
| i | Britten | England | 11/22/1913 | 12/4/1976 |
| | Buxtehude | Germany | 16/1688 | 16/1688 |