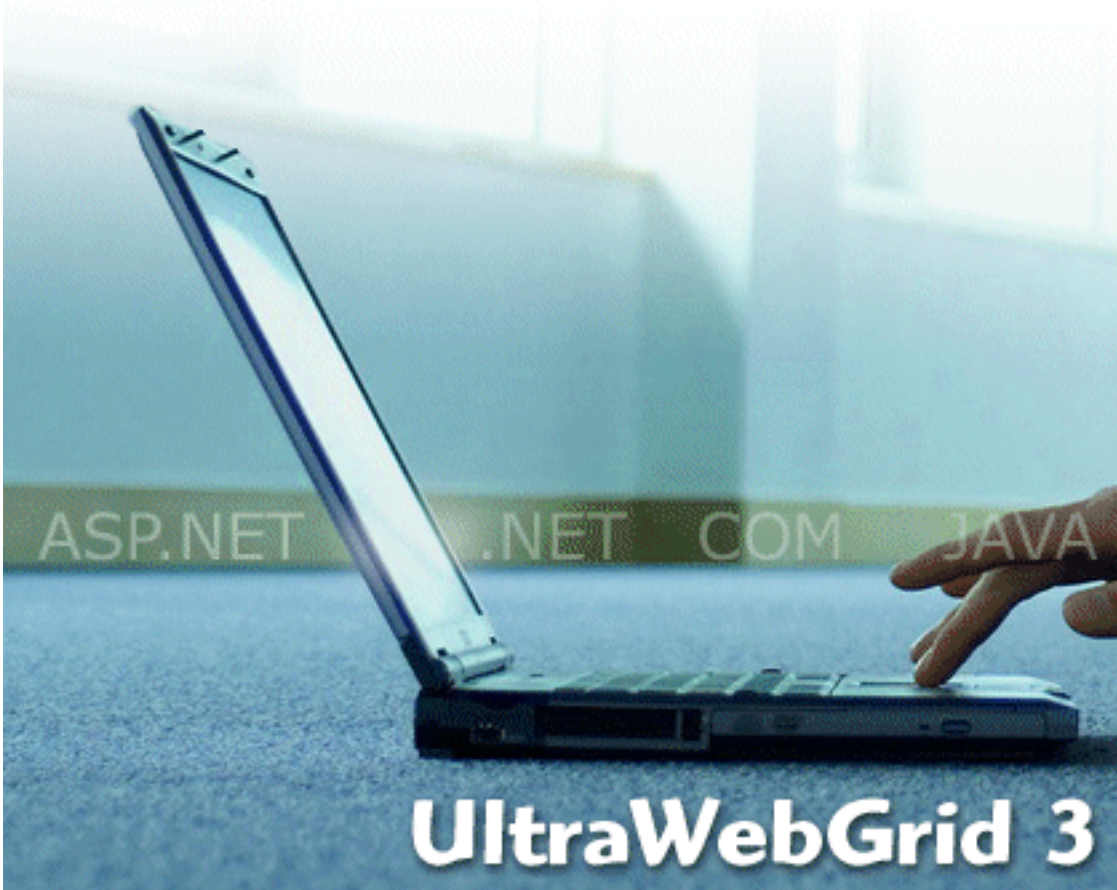




Volume 2004.2



Infragistics.WebUI.UltraWebGrid

 [Title Page](#)

 Understanding WebGrid

 [Documentation Atlas](#)

 [WebCombo Client-Side Object List](#)

 [WebGrid Client-Side Object List](#)

 [Section 508 Compliance](#)

 How To Deploy WebGrid

 [Setting CopyLocal to True](#)


 [Deploying Projects with UltraWebGrid](#)

 [Distributable Files](#)

 Introductions To WebGrid Features

 [ViewState](#)

 [Keeping ViewState In SessionState](#)

 [Tabular Display and the ViewType Property](#)

 [Cross-Browser Compatibility](#)

 [DisplayLayout](#)

 [Style Inheritance](#)

 [Client Side Object Model](#)

 [Events](#)

 [Data Binding](#)

 [Excel Exporting](#)

 [LoadOnDemand](#)

 [Outlook GroupBy mode](#)

 [Bands](#)

 [Rows](#)

 Design-Time Utilities

 [The Auto-Format Wizard](#)

 [The Property Builder](#)

 [Advanced JavaScript Architecture](#)

 [Objects Created from Rendered HTML and JavaScript](#)










 [Objects Created on the Client Dynamically](#)

 [Use the DOM To Navigate Rows-Columns-Cells](#)

 Revision History

 [New For Version 2.1](#)

 [What's New In Version 2](#)

-  [The WebCombo Control](#)
-  [Using Read-Only Mode](#)
-  [Loading and Saving layouts](#)
-  [Row and Column Templates](#)
-  [Using Hidden Columns](#)
-  [Column Moving and Sorting](#)
-  [Hiding Text Overflow and Displaying Ellipsis](#)
-  [Cell Merging](#)
-  [Expanded Help File](#)

Revision History

-  [What's New in Version 3](#)
-  [Release Notes](#)

WebGrid Client-Side Object Tree

-  [Introduction and Overview](#)

-  [Utility Functions](#)

Grid Object

-  [Bands Property](#)

-  [Band Object](#)

-  [Row Object](#)

Band Object

-  [Columns Property](#)

-  [Column Object](#)

Column Object

-  [ValueList Property](#)

Row Object

-  [Rows Collection \(Child Rows\)](#)

Rows Collection

-  [Row Object](#)

-  [Cell Object](#)

-  [ExpandEffects Object](#)

Task-Based Help

Server-Side Tasks

Data Sources

Bound Data

-  [DataSet](#)

-  [Data Table](#)

-  [XML](#)

-  [Custom Data Class](#)
-  [Collection Of Objects](#)
-  [Use Data Binding Events](#)
-  [Filter Columns From The Datasource](#)




Hierarchical Data

-  [DataSet Relation](#)

-  [Use In-Memory Dataset for ValueList](#)

Data Manipulation & Validation

Updating Data

-  [Data Validation on the Server](#)
-  [Data Validation on the Client](#)
-  [Updating with a Multi-Column Primary Key](#)

-  [Create an Unbound Column](#)

-  [LoadOnDemand](#)

-  [Set Up Outlook GroupBy in Code](#)

-  [Handling Sorting Events](#)

-  [Handling Paging Events](#)

-  [Display One Value But Store Another with ValueLists](#)

-  [Use In-Memory Dataset for ValueList](#)

Styles and Formatting

-  [Using Styles To Change Grid Appearance](#)

-  [Create and Apply Styles](#)

-  [Using CSS StyleSheets in the Grid](#)

-  [Loading and Saving Layouts](#)

-  [Row and Column Templates](#)

-  [Set the Border Style on the Active Row](#)

-  [Set Colors for Alternate Rows](#)

-  [Change the Height of a Single Row](#)

-  [Hide Row Selectors](#)

-  [Stationary Column Headers - Footers](#)

-  [Hide Cell Text Overflow and Display Ellipsis](#)

-  [Display A Picture In A Grid Cell](#)

-  [Change Cell Style Based On Value](#)

-  [Cell Merging](#)







-  [Set Cell Borders and Grid Lines](#)

User Interface: Selection-Activation-Editing





-  [Using the Activation Object](#)

-  [Row and Column Templates](#)
-  [Using Read-Only Mode](#)
-  [Removing Scrollbars Entirely](#)
-  [Row Selection and ActiveRow](#)
-  [Change the Active Row](#)
-  [Expand All Rows in the Grid](#)
-  [Scroll a Row Into View](#)
-  [Move \(Re-position\) Columns](#)
-  [Cell Selection and Active Cell](#)
-  [Prevent Editing of Certain Cells](#)
-  [Begin Editing a Cell When Clicked](#)
-  [Embedding WebTextEditors](#)
-  [Add a Custom Editor to a Column](#)
-  [Mult-Line Cell Editing](#)
-  [Display One Value But Store Another with ValueLists](#)
-  [Value Lists that Change from Row to Row](#)






Row-Column-Cell Handling (Non-UI)

-  [Loop Through the Rows in a Band](#)
-  [Adding Rows Without Default Values](#)
-  [Adding Rows Without an AutoNumber Field](#)
-  [Return a Specific Cell's Value](#)
-  [Updating a Column Footer](#)
-  [Set Up Outlook GroupBy in Code](#)

Using Server-Side Events

-  [Configure the Grid with InitializeLayout](#)
-  [Using InitializeRow](#)
-  [Handling Sorting Events](#)
-  [Handling Paging Events](#)

Working With ValueLists

-  [Get the Newly Selected Item in a ValueList](#)
-  [Display One Value But Store Another with ValueLists](#)
-  [Use In-Memory Dataset for ValueList](#)
-  [Value Lists that Change from Row to Row](#)
-  [WebCombo With ValueList](#)

WebCombo

-  [WebCombo With ValueList](#)

Client-Side Tasks

Using Client-Side Events

 [Handle Client-Side Events](#)

 [How the Event Model Functions](#)

ClientSideEvents Class

 [Overview](#)

 [Members](#)

 [Trigger a Postback Within a Client-Side Event](#)

 [How To Get a Band From a Row ID](#)

 [Object Looping](#)

 [Suspend Updates](#)

Data Manipulation

 [Deleting Rows Using DeleteRows](#)

 [Updating Cells On Client](#)

 [Using AddNew to Add a Row](#)

 [Using Hidden Columns](#)

 [Retrieving Cell Values for a Hidden Column](#)

Formatting and Appearance

 [Change Row Height](#)

 [Making Headers and Footers Stationary](#)

 [Change Column Width](#)

 [Change The Header Text of a Column](#)

 [Update A Column Footer](#)

 [Hiding the AddNew Box](#)

 [Change Grid Formatting Through Code](#)

 [Change the Scrollbar Color](#)

Interacting with The User

 [Placing a Cell in Edit Mode](#)

 [How To Cancel Certain Keystrokes](#)

 [Change the Value of a Cell in Response to User Edit](#)

 [Expand and Collapse Rows on the Client-Side](#)

 [Prevent the Deletion of Certain Rows on Client](#)

 [Handling Edit Mode](#)

 [Confirm Delete With Message Box](#)

 [Searching \(Find and Find Next\)](#)

 [Popup a WebMenu on Right-Click](#)

Selection-Sorting-Activation

 [Select Rows Using SelectRow](#)

 [How To Set an ActiveRow](#)

 [Postback to Sort By a Different Column](#)

 [De-Select All Rows At Once](#)

 [Select Rows-Columns-Cells on Client-Side](#)

 [Column Moving and Sorting](#)

 WebCombo


 [Manually Drop Down WebCombo](#)

 [WebCombo As DropDown List](#)

 JavaScript Files

 [Change JavaScript Files](#)

 [Tutorials: Learn To Use WebGrid/Combo](#)

 Introductory

 [Quick Start - Bound Mode \(Flat\)](#)

 [Quick Start - Bound Mode \(Hierarchical\)](#)

 [Quick Start - Unbound Mode](#)

 [Create a Grid Dynamically](#)

 [Additional Code Resources](#)

 In-Depth

 [Quick Start](#)

 [Bind to Northwind Flat](#)

 [Bind to Northwind Hierarchical](#)

 [Initialize Layout](#)

 [Initialize Row](#)

 [Band Appearance](#)

 [Row Appearance](#)

 [Row Alternate Appearance](#)

 [Row Templates](#)

 [Column Appearance](#)

 [Column Moving](#)

 [Stationary Column Headers](#)

 [Column Sorting and Sort Indicators](#)

 [Column Templates](#)









 [Cell Appearance](#)

 [Cell Merging](#)

 [Cell Formatting](#)

 [Cell Validation](#)

 [Web Combo in Grid Cell](#)

-  [Outlook GroupBy](#)
-  [Read-Only Mode](#)
-  [Exporting To Excel](#)
-  [Save and Load XML Layout](#)
-  [Create WebGrid With Code](#)
-  [Update Database With Row Changes](#)
-  [Update Database In Batch Mode](#)
-  [Data Paging \(Basic\)](#)
-  [Data Paging \(Advanced\)](#)
-  [Load On Demand](#)
-  [Multiple Data Key Fields](#)
-  [Program On Client-Side \(Quick Start\)](#)
-  [Program On Client-Side \(Introduction\)](#)
-  [Program On Client-Side \(Intermediate\)](#)
-  [Program On Client-Side \(Advanced\)](#)
-  [Server Events](#)
-  [WebGrid on User Control](#)
-  [WebCombo On A Form](#)

API Reference

Namespaces

Infragistics.WebUI.UltraWebGrid Namespace

[Overview](#)

Classes

[ActivationObject](#)

[Members](#)

[AddNewBox](#)

[Members](#)

[BandEventArgs](#)

[Members](#)

[BandsCollection](#)

[Members](#)

[BrowserChangedEventArgs](#)

[Members](#)

[CellEventArgs](#)

[Members](#)

[CellItem](#)

[Members](#)

 [CellsCollection](#)

 [Members](#)

 [ChangedCellPair](#)

 [Members](#)

 [ClickEventArgs](#)

 [Members](#)

 [ClientSideEvents](#)

 [Members](#)

 [ColumnControlIDEditor](#)

 [Members](#)

 [ColumnControlIDEditor.ConsumerImpl](#)

 [Members](#)

 [ColumnDataType](#)

 [Members](#)

 [ColumnEventArgs](#)

 [Members](#)

 [ColumnsCollection](#)

 [Members](#)

 [EditorControlIDEditorBase](#)

 [Members](#)

 [ExpandEffects](#)

 [Members](#)

 [FooterEventArgs](#)

 [Members](#)

 [FooterItem](#)

 [Members](#)

 [GridItemStyle](#)

 [Members](#)

 [GroupByBox](#)

 [Members](#)

 [GroupByRow](#)

 [Members](#)




















 [HeaderItem](#)

 [Members](#)

 [ImageUrls](#)

 [Members](#)

 [InGridRenderer](#)

-  [Members](#)
-  [InvalidCellsEnumerator](#)
-  [Members](#)
-  [LayoutEventArgs](#)
-  [Members](#)
-  [PageEventArgs](#)
-  [Members](#)
-  [Pager](#)
-  [Members](#)
-  [RendererConsumerBase](#)
-  [Members](#)
-  [RowEventArgs](#)
-  [Members](#)
-  [RowsCollection](#)
-  [Members](#)
-  [RowsEditor](#)
-  [Members](#)
-  [SelectedCellsCollection](#)
-  [Members](#)
-  [SelectedCellsEventArgs](#)
-  [Members](#)
-  [SelectedColsCollection](#)
-  [Members](#)
-  [SelectedColumnsEventArgs](#)
-  [Members](#)
-  [SelectedRowsCollection](#)
-  [Members](#)
-  [SelectedRowsEventArgs](#)
-  [Members](#)
-  [SortColumnEventArgs](#)
-  [Members](#)
-  [SortedColsCollection](#)
-  [Members](#)
-  [SpecialBoxBase](#)
-  [Members](#)
-  [Strings](#)
-  [Members](#)

 [TemplatedColumn](#)

 [Members](#)

 [UltraGridBand](#)

 [Members](#)

 [UltraGridCell](#)

 [Members](#)

 [UltraGridColumn](#)

 [Members](#)

 [UltraGridEventArgs](#)

 [Members](#)

 [UltraGridLayout](#)

 [Members](#)

 [UltraGridRow](#)

 [Members](#)

 [UltraGridRowsEnumerator](#)

 [Members](#)

 [UltraWebGrid](#)

 [Members](#)

 [UpdateEventArgs](#)

 [Members](#)

 [ValidatorItem](#)

 [Members](#)

 [ValidatorItemsCollection](#)

 [Members](#)

 [ValueList](#)

 [Members](#)

 [ValueListItem](#)


 [Members](#)


 [ValueListItemsCollection](#)

 [Members](#)

 [XmlClientWrite](#)

 [Members](#)

 [Enumerations](#)

 [Structures](#)

 [ProcessRowParams](#)

 [Members](#)

 [Interfaces](#)

 [ICanEditNavBar](#)

 [Members](#)

 [IControlEditorConsumer](#)

 [Members](#)

 [IPlugInConsumer](#)


 [Members](#)

 [IPlugInFactory](#)

 [Members](#)

 [IPlugInRender](#)

 [Members](#)

 [IResolveStyles](#)

 [Members](#)


 [IUltraGridExporter](#)

 [Members](#)

 [Delegates](#)

 [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

 [Overview](#)

 [Classes](#)

 [BeginExportEventArgs](#)

 [Members](#)

 [CellExportedEventArgs](#)

 [Members](#)

 [CellExportingEventArgs](#)

 [Members](#)

 [ColumnSet.ColumnIndex](#)

 [Members](#)

 [EndExportEventArgs](#)

 [Members](#)

 [ExcelExportCancelEventArgs](#)

 [Members](#)

 [ExcelExportEventArgs](#)

 [Members](#)

 [ExcelExportInitializeRowEventArgs](#)

 [Members](#)

 [HeaderCellExportedEventArgs](#)

 [Members](#)

 [HeaderCellExportingEventArgs](#)

 [Members](#)

 [HeaderRowExportedEventArgs](#)

 [Members](#)

 [HeaderRowExportingEventArgs](#)

 [Members](#)

 [InitializeColumnEventArgs](#)

 [Members](#)

 [RowExportedEventArgs](#)

 [Members](#)

 [RowExportingEventArgs](#)

 [Members](#)

 [SummaryCellExportedEventArgs](#)

 [Members](#)

 [SummaryCellExportingEventArgs](#)

 [Members](#)

 [SummaryRowExportedEventArgs](#)


 [Members](#)

 [SummaryRowExportingEventArgs](#)

 [Members](#)

 [UltraWebGridExcelExporter](#)

 [Members](#)

 [Enumerations](#)

 [Interfaces](#)


 [IGenerateGridExportEvents](#)

 [Members](#)

 [Delegates](#)

 [Infragistics.WebUI.WebCombo Namespace](#)

 [Overview](#)

 [Classes](#)

 [ClientSideEvents](#)

 [Members](#)

 [ColumnsListEditor](#)

 [Members](#)

 [DropDownLayout](#)

 [Members](#)

 [ExpandEffects](#)


 [Members](#)


 [SelectedRowChangedEventArgs](#)

 [Members](#)

 [WebCombo](#)

 [Members](#)

 Enumerations

 Interfaces

 [ICanDropDown](#)

 [Members](#)

 Delegates


Documentation Atlas

The Documentation Atlas topic provides several high-level overviews (maps) of the UltraWebGrid help system. You can use it to quickly find information that is relevant to your concern, or to learn how to use the help system for maximum benefit.

This topic also contains the following sections:

- [Main Sections Of The Help System](#)
- [Control Properties and Events](#)
- [Top-Level Object List](#)
- [Other Useful Resources](#)

Note To use the help system effectively, you must be able to see the Table Of Contents pane. The TOC can be displayed from within Visual Studio by selecting the Help menu and choosing Contents, or by pressing **Ctrl-Alt-F1** on your keyboard. Once the TOC is displayed, find the top level "book" node that matches the name of your product (Infragistics.WebUI.UltraWebGrid). Double-click it or press its "+" symbol to expand its children. Everything under this node is part of the UltraWebGrid help system.

You can also open the UltraWebGrid help system at the top level in a separate help viewer window. To do this, select the Windows Start Menu, choose Programs (All Programs under Windows XP) and then choose Infragistics. You will see a sub-menu with the name of your product (UltraWebGrid). Select this option, and you will see a  icon labelled UltraWebGrid Help. Select this item to open the help viewer.

Main Sections of the Help System

The help system for the UltraWebGrid is divided into several main areas. Each area provides a different way for you to learn about the controls. The main areas of the help file appear as top-level nodes in the Table Of Contents (TOC) pane for the help file.

The main sections of the help system are:

Section	Description
What's New In Version 3	This section contains overview descriptions of the new features that have been added to this version of the product.
Introductions and Important Information	<p>This section contains several sub-sections and two very important topics.</p> <p>The two top-level topics cover setting CopyLocal to True (required to test your application during development) and deploying your UltraWebGrid application. These topics are very important and should be reviewed by every developer who is using the product.</p> <p>The first sub-section, Introductions to WebGrid Features, presents a series of introductory summaries of the product's main features. Understanding the features in this section is critical to understanding how the controls work. These topics also highlight various benefits of using the product.</p> <p>The second sub-section, Design-Time Utilities, contains descriptions of the design-time utilities included with the product. Use this section to gain a better understanding of the design-time tools provided to assist you in setting up the UltraWebGrid in your application.</p>

The third sub-section, **Advanced JavaScript Architecture**, covers important information for developers who wish to use the client-side features of the control to their full potential. Its topics describe the implementation of the products' JavaScript architecture for client-side operations. *You can click the top-level node of this sub-section in the TOC to view a summary topic.*

The **Revision History** sub-section contains the "What's New" topics from previous versions of the product.

Tutorials: Learning To Use WebGrid/Combo

The Tutorials section provides guided learning through a series of illustrated exercises that show you how to use the control. There are two sub-sections. The **Introductory** sub-section provides a simple overview of just the main tasks involved with getting the WebGrid up and running in your application. It also includes a topic with links to additional code resources available from the Infragistics web site.

The **In Depth** sub-section provides a much more detailed and exhaustive set of exercises that cover most aspects of the product, from simple data binding to advanced client-side functionality.

Solve Common Problems (Task-Based Help)

While the purpose of Tutorials is to teach you how to use the product through exercises that illustrate the control's features, the Task-Based Help aims to aid you in quickly accomplishing a specific task. Use Task-Based Help when you just want to know how to accomplish a particular goal.

Briefly, the Tutorials focus on the product and on showing all that it can do. The Task-Based Help focuses on you and on what you want to accomplish right now.

To use Task-Based Help, first think about what it is you are trying to do, then choose whether you want to accomplish the task using server-side or client-side functionality. Select the appropriate sub-section. Next, decide which of the available categories most likely covers the type of activity you are interested in, and expand that category. Finally, select the topic that most closely matches your interest. If your intended activity seems to fit equally well into multiple categories, try any of them; some topics are listed multiple times under different categories.

Object Trees

The Object Trees are structures that appear in the Table Of Contents. They illustrate the hierarchical relationships of the various objects that make up the controls. You can use them to see how objects relate to one another and to determine how you should access a particular type of object. Clicking on any object in the tree takes you to a help topic covering that object. Some of the objects also have separate overview topics available.

There are separate object trees for the client-side object model and the server-side object model. Select the one you want based on the type of programming you are doing (JavaScript vs. Visual Basic / C#).

Namespace API Reference

The API Reference comprises many detailed topics that cover every aspect of the control. All the classes used by the assemblies are listed, with all of their properties, methods, events, enumerations, etc. Use this section of the help system to get highly-specific information about the control's features.

API reference topics include overview and summary information, syntax for supported languages, hierarchy diagrams, code samples, links to related information, and more.

Control Properties and Events

The controls themselves appear in the API reference hierarchy as objects, so to locate a particular control, you must first expand it's assembly namespace, expand the Classes node, then locate the class that matches the name of the control, either UltraWebGrid or WebCombo.

UltraWebGrid

- To see the UltraWebGrid's Overview topic from the API refrence, [click here](#).
- For a list of all the properties, methods, events, etc. of the UltraWebGrid, [click here](#).

UltraWebCombo

- To see the UltraWebCombo's Overview topic from the API refrence, [click here](#).
- For a list of all the properties, methods, events, etc. of the UltraWebCombo, [click here](#).

Top-Level Object List

The following are the top-level objects found in the UltraWebGrid Controls. This list excludes collections, enumerations, event arguments and other topics that appear in the API Reference portion of the TOC.

Note Object names in italics are *aliases* for the actual object, which also appears under its correct name. For example, *Band* is an alias for the *UltraGridBand* object.

UltraWebGrid

- | | | |
|--|---|--|
| <ul style="list-style-type: none">• ActivationObject• AddNewBox• Band• BorderDetails• Cell• CellItem• ChangedCellPair• ClientSideEvents• Column• DisplayLayout• ExpandEffects• FooterItem | <ul style="list-style-type: none">• GroupByBox• GroupByRow• HeaderItem• ImageURLs• Layout• Margin• Padding• Pager• Rectangle• Row• SpecialBoxBase• Style | <ul style="list-style-type: none">• UltraGridBand• UltraGridCell• UltraGridColumn• UltraGridLayout• UltraGridRow• UltraGridStyle• UltraWebGrid• ValidatorItem• ValueList• ValueListItem• WebGrid control |
|--|---|--|

- [GridItemStyle](#)
- [TemplatedColumn](#)

UltraWebCombo

- [ClientSideEvents](#)
- [DropDownLayout](#)
- [ExpandEffects](#)
- [WebCombo](#)

Other Useful Resources

The following topics are other resources found in the help system that may address questions or problems you have regarding the product or its documentation.

- [Client-Side Object Model Introduction](#) - This topic gives a generalized overview of the Client-Side Object model, plus it contains links to all the main topics that document the properties, methods and events of the client-side object model. If you are doing client-side programming, this topic should be your first stop.
- [Additional Code Resources](#) - This topic provides links to the Knowledge Base section of the Infragistics web site. You can view articles and download commented source code for projects that illustrate key UltraWebGrid concepts. Note that a number of articles you may find in the Knowledge Base for this product have already been converted into Task-Based Help topics.

Client-Side Object Model - Combo Object

The Combo Object of UltraWebCombo can be obtained within the page using the `igcmbo_getComboById` utility function.

```
var combo = igcmbo_getComboById('UltraWebCombo1');
```

The combo can also be referenced directly through the definition that is automatically generated for it. If the name of the combo on the server is 'UltraWebCombo1' then a variable called `oUltraWebCombo1` will be available from anywhere within the page. The 'o' prefix is a convention used to indicate that the variable is an object as opposed to an HTML element. When a variable represents an HTML element, an 'e' is prefixed in front of it.

This topic also contains the following sections:

- [Methods](#)
- [Events](#)

Properties

CancelPostBack

Gets or sets a Boolean value that indicates whether postback should be cancelled when the next event occurs. If a postback is set to occur due to some other client-side activity, setting this property to True will prevent the postback from occurring. This property can be changed on the client.

```
var combo = igcmbo_getComboById('UltraWebCombo1');  
combo.CancelPostBack = true;
```

DataTextField

Identifies the column key value used as the source of display text in the top portion of the WebCombo. This property corresponds to the property of the same name on the server. This property is read-only and should not be changed on the client.

DataValueField

Identifies the column key value used as the source of data values posted back to the server for WebCombo. This property corresponds to the property of the same name on the server. This property is read-only and should not be changed on the client.

Editable

Returns a Boolean value that indicates whether or not the WebCombo element is editable in the top portion of the element. If true, the user can enter their own values in the combo, if false they user is restricted to selecting values already present in the dropdown. This property corresponds to the property of the same name on the server. This property is read-only and should not be changed on the client.

```
var combo = igcmbo_getComboById('UltraWebCombo1');  
var canBeEdited = combo.Editable;
```

Element

Returns the HTML element object corresponding to the WebCombo on the page. This property is read-only and should not be changed.

```
var element = igcmbo_getComboById('WebCombo1').Element; element.style.color = "red";
```

Events

[See below for Events.](#)

ExpandEffects

This property returns an object that contains the values set for the expand effects that will be supplied when WebCombo is dropped down. For more detail, see the [ExpandEffects object topic](#). The ExpandEffects object

contains the same properties as the server-side version. This object corresponds to the property of the same name on the server.

```
var expfx = igcmbo_getComboById('WebCombo1').ExpandEffects; expfx.Duration = 100;
expfx.Opacity = 70;
```

Id

Returns the HTML id property of the top-level WebCombo element on the page. This property is read-only and should not be changed.

```
var id = oUltraWebCombo1.Id;
```

NeedPostBack

Gets or sets a Boolean value that indicates whether a postback should be initiated after the following event occurs. Set this property to True to force a manual postback to occur after the next event. This property can be changed on the client.

```
oWebCombo.NeedPostBack = true;
```

Methods

getDisplayValue()

Returns the current value of the top portion of the Webcombo element.

Parameters:

None.

```
var value = oCombo.getDisplayValue();
```

getDropDown()

Returns a boolean value that indicates whether or not the WebCombo dropdown is currently displayed.

Parameters:

None.

```
if(oCombo.getDropDown())
    return;
else
    oCombo.setDropDown(true);
```

getGrid()

Returns a reference to the oGrid object that manages the dropdown area of the WebCombo element. This object supports the entire object and programming model of the WebGrid element on the client.

Parameters:

None.

```
var oGrid = oCombo.getGrid();
// Turn off sorting in the dropdown
oGrid.AllowSort = 2;
```

getSelectedIndex()

Returns the index offset of the currently selected row of the dropdown grid.

Parameters:

None.

```
var index = oCombo.getSelectedIndex();
```

setVisible()

Returns a boolean value that indicates whether or not the WebCombo element is visible on the page.

Parameters:

None.

```
var visible = oCombo.setVisible();
if(!visible)
    oCombo.setVisible(true);
```

setDisplayValue(newValue, bFireEvent)

Sets the value and the text in the top portion of the WebCombo element to the passed in *newValue* parameter. The second parameter, *bFireEvent*, is a boolean value that indicates whether **BeforeSelectChange** and **AfterSelectChange** events should be fired on the client.

Parameters:

newValue- A string value that will be assigned to the element.

bFireEvent- A Boolean value that specifies whether the **BeforeSelectChange** and **AfterSelectChange** client-side events should be fired. If true, the events will occur.

```
oCombo.setDisplayValue("John Doe", true);
```

setDropDown(bDrop)

This method controls the dropdown state of WebCombo. Passing true as a parameter causes the dropdown to display if it is not already displayed. Passing false causes the dropdown to disappear if it is visible.

Parameters:

bDrop- A Boolean value that specifies the dropped-down state of the element.

```
oCombo.setDropDown(true);
```

setVisible(bVisible)

Sets the visibility of the WebCombo according to the passed-in Boolean value. If true, the WebCombo element is displayed at its current position. If false, the WebCombo element is hidden on the page.

Parameters:

bVisible- A Boolean value that specifies the visibility of the WebCombo element.

```
oCombo.setVisible(false);
```

setWidth(width)

Sets the width of the WebCombo element in pixels.

Parameters:

width- An integer value that specifies the width of the element.

```
oCombo.setWidth(150);
```

Events

The **Events** object contains over 25 members that are used to specify JavaScript function handlers that are called in response to a variety of events occurring on the client. The **Event** member is only populated with a valid JavaScript function if the function name is set from the **Events** object of the **DisplayLayout** object on the server.

All events are called with at least one parameter: The ID of the WebCombo on which the event occurred. In the case of the **EditKeyDown** and **EditKeyUp** events, additional parameters are added which specify the current or new value for the edit portion of the combo, and the keycode of the keystroke that caused the event.

The members of the **Events** object are:

- AfterCloseUp(id)
- AfterDropDown(id)
- AfterSelectChange(id)
- BeforeCloseUp(id)
- BeforeDropDown(id)
- BeforeSelectChange(id)
- EditKeyDown(id, currentValue, keyCode)
- EditKeyUp(id, newValue, keyCode)
- InitializeCombo(id)

Client-Side Object Model - Grid Object

The Grid Object of UltraWebGrid can be obtained within the page using the `igtbl_getGridById` utility function.

```
var grid =igtbl_getGridById('UltraWebGrid1');
```

The grid can also be referenced directly through the definition that is automatically generated for it. If the name of the grid on the server is 'UltraWebGrid1' then a variable called `oUltraWebGrid1` will be available from anywhere within the page. The 'o' prefix is a convention used to indicate that the variable is an object as opposed to an HTML element. When a variable represents an HTML element, an 'e' is prefixed in front of it.

This topic also contains the following sections:

- [Methods](#)
- [Events](#)

Properties

AddNewBoxView

Gets an integer value that indicates what type the AddNewBox object is. This property is read-only and should not be changed.

0 = Full

1 = Compact

AddNewBoxVisible

Gets a Boolean value that indicates if the AddNewBox is visible for the grid. This property is read-only and should not be changed.

AllowAddNew

Gets or sets a value that indicates if the adding rows is permitted in the grid. This property can be changed on the client. The `alignGrid` method has to be called after changing the property value in order to reflect changes on the page.

0 - Not Set

1 = Yes

2 = No

```
var grid = igtbl_getGridById('UltraWebGrid1');
```

```
if(grid.AllowAddNew == 1)
```

```
{
```

```
    igtbl_getElementById("MyLabel").innerHTML = "Press the button to add a row";
```

```
}
```

AllowColSizing

Gets or sets a value that indicates if column sizing is allowed for the grid as a whole. This property can be changed on the client.

0 - NotSet

1 = Fixed

2 = Free

```
var grid = igtbl_getGridById('UltraWebGrid1');
```

```
grid.AllowColSizing = 1;
```

AllowDelete

Gets or sets a Boolean value that indicates if row deleting is allowed for the grid as a whole. This property can be changed on the client.

0 - Not Set

1 = Yes

2 = No

```
var grid = igtbl_getGridById('UltraWebGrid1');
```

```
grid.AllowDelete = 2;
```

AllowPaging

Gets a Boolean value that indicates whether grid data is being displayed via paging. If paging is being used to display data, you can use the **PageCount** and **CurrentPageIndex** properties to manage paging on the client-side. This property is read-only and should not be changed.

AllowSort

Gets a value that indicates if sorting is in effect for the grid. This property is read-only and should not be changed.

0 - Not Set
1 = Yes
2 = No
3 = OnClient

AllowUpdate

Gets or sets a value that indicates if cell editing is allowed for the grid as a whole. You can also use this property to specify that editing is allowed only using row templates, which disables the in-place editing of grid data. This property can be changed on the client. But be careful, if you're going to turn it on/off on the client make sure that you set it to yes on the server and then set on the client to anything you want. Otherwise the HTML code for the edit box won't be rendered down.

0 - Not Set
1 = Yes
2 = No
3 = RowTemplateOnly

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.AllowUpdate = 1;
```

AltClass

Gets or sets a CSS class name value that specifies the style for alternate cells in the grid. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.AltClass = "MyAlternateCssClass";
```

Bands

Gets the array of Band objects for the grid. The length of this array is equal to the number of bands in the grid hierarchy. There is always at least one Band object in the array.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.bands[0].RowSizing = 2;
```

BlankImage

Gets or sets a string value that determines the image displayed to represent a blank space in the row selector. The purpose of this image is to create proper spacing in the browser. This property can be changed on the client.

```
var grid = igtbl_getGridById('G_UltraWebGrid1');
grid.BlankImage = "MyBlankImage.gif";
```

CancelPostBack

Gets or sets a Boolean value that indicates whether postback should be cancelled when the next event occurs. If a postback is set to occur due to some other client-side activity, setting this property to True will prevent the postback from occurring. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.CancelPostBack = true;
```

CaseSensitiveSort

The property is used on the client to determine whether the client side sorting should consider the letters' case during the sorting procedure. Default value is false. Needs to be set anytime before client side sorting is done to switch between case sensitive/unsensitive sort. Has no analog on the server side.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.CaseSensitiveSort = true;
```


CellClickAction

Gets or sets an integer value that indicates what action will be taken when a cells are clicked in the grid by default. The property can be changed on the client.

1 = Edit
2 = Row Select
3 = Cell Select

CollapseImage

Gets or sets a string value that determines the image displayed to denote the collapsing of an expanded row. This property can be changed on the client. New image will appear on the newly added or recently expanded rows only.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.CollapseImage = "MyCollapseImage.gif";
```

CurrentPageIndex

Gets an integer value that specifies the index of the page of grid data that is being displayed. This property is only useful when data is being displayed in multiple pages. You can determine whether data paging is being used by checkin the value of the **AllowPaging**. This property is read-only and should not be changed.

CurrentEditRowImage

Gets or sets a string value that determines the image displayed when a row is the current, active row and it can be modified using its row template by clicking on the image. This property can be changed on the client. New image will appear after changing the current row.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.CurrentEditRowImage = "MyCurrentEditRowImage.gif";
```

CurrentRowImage

Gets or sets a string value that determines the image displayed when a row is the current, active row. This property can be changed on the client. New image will appear after changing the current row.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.CurrentRowImage = "MyCurrentRowImage.gif";
```

EditCellClass

Gets or sets a CSS class name value that specifies the style for cells in the grid that are being edited. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.EditCellClass = "MyEditCssClass";
```

Element

Returns the HTML element object corresponding to the WebGrid on the page. This property is read-only and should not be changed.

```
var element = igtbl_getGridById('UltraWebGrid1').Element;  
element.style.color = "red";
```

Events

[See below for Events.](#)

Expandable

Gets an integer value that indicates if rows are expandable on the page. This property is read-only and should not be changed.

1 = Yes
2 = No

ExpandImage

Gets or sets a string value that determines the image displayed to denote the expansion of a collapsed. This property can be changed on the client. New image will appear on the newly added or recently collapsed rows only.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.ExpandImage = "MyExpandImage.gif";
```

ExpAreaClass

Gets or sets a CSS class name value that specifies the style for the row expansion areas of the grid. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.ExpAreaClass = "MyExpCssClass";
```

FooterClass

Gets or sets a CSS class name value that specifies the style for footer cells in the grid. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.FooterClass = "MyFooterCssClass";
```

GroupByRowClass

Gets or sets a CSS class name value that specifies the style for GroupBy rows in the grid. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.GroupByRowClass = "MyGroupByCssClass";
```

HeaderClass

Gets or sets a CSS class name value that specifies the style for Column Headers in the grid. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.HeaderClass = "MyHeaderCssClass";
```

HeaderClickAction

Gets or sets an integer value that indicates what action will be taken when a column headers are clicked in the grid by default. The property can be changed on the client.

```
0 = NotSet
1 = Select
2 = SortSingle
3 = SortMulti
```

Id

Returns the HTML id property of the top-level WebGrid element on the page. This property is read-only and should not be changed.

```
var id = oUltraWebGrid1.Id;
```

Indentation

Gets the number of pixels used for indenting one band of rows from another in the grid. This property is read-only and should not be changed.

ItemClass

Gets or sets a CSS class name value that specifies default style for cells in the grid. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.ItemClass = "MyCssClass";
```

NeedPostBack

Gets or sets a Boolean value that indicates whether a postback should be initiated after the following event occurs. Set this property to True to force a manual postback to occur after the next event. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.NeedPostBack = true;
```

NewRowImage

Gets or sets a string value that determines the image displayed to denote a row that has been added. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.NewRowImage = "MyNewRowImage";
```

NullText

Gets or sets a string value that determines the text displayed to denote a null value in a cell. This property can be changed on the client. Be aware that setting it on the client won't change all the null values in the grid right away, it will be applied to changed and newly added cells only.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.NullText = "[no value]";
```

PageCount

Gets or sets an integer value that specifies the number of pages being used to display grid data. This property is read-only and should not be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
var numpages = grid.PageCount;
```

RowLabelClass

Gets or sets a CSS class name value that specifies the style for row selector labels in the grid. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.RowLabelClass = "MyRowLabelCssClass";
```

Rows

This property returns all the Row objects for Band 0 in the grid. You can use this property to access individual rows in Band 0 and from there gain access to the other bands of the grid.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.rows[0].RowSizing = 2;
```

RowSelectors

Gets a Boolean value that indicates if rows selectors are visible on the page. This property is read-only and should not be changed on the client.

1 = Yes
2 = No

RowSizing

Gets an integer value that indicates if row sizing is allowed for the grid as a whole. This property can be changed on the client.

1 = Fixed
2 = Free

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.RowSizing = 1;
```

SelCellClass

Gets or sets a CSS class name value that specifies the style for selected cells in the grid. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.SelCellClass = "MySelCellCssClass";
```

SelectTypeCell

Gets or sets an integer value that indicates the type of cell selection that is in effect for the grid. This property can be changed on the client.

0 = NotSet
1 = None
2 = Single
3 = Extended

```
var grid = igtbl_getGridById('UltraWebGrid1');
grid.SelectTypeCell = 2;
```

SelectTypeColumn

Gets or sets an integer value that indicates the type of column selection that is in effect for the grid. This property can be changed on the client.

0 = NotSet

1 = None

2 = Single

3 = Extended

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.SelectTypeColumn = 2;
```

SelectTypeRow

Gets or sets an integer value that indicates the type of row selection that is in effect for the grid. This property can be changed on the client.

0 = NotSet

1 = None

2 = Single

3 = Extended

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.SelectTypeRow = 2;
```

SelGroupByRowClass

Gets or sets a CSS class name value that specifies the style for selected GroupBy rows in the grid. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.SelGroupbyRowClass = "MySelGroupByCssClass";
```

SelHeadClass

Gets or sets a CSS class name value that specifies the style for selected column headers in the grid. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.SetHeadClass = "MySelHeadCssClass";
```

ShowBandLabels

Gets or sets a Boolean value that indicates if band labels are visible on the page.

SortAscImg

Gets or sets a string value that determines the image displayed in the column header of a column that is sorted in ascending order. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.SortAscImg = "AscSortArrow.gif";
```

SortDscImg

Gets or sets a string value that determines the image displayed in the column header of a column that is sorted in descending order. This property can be changed on the client.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.SortDscImg = "DscSortArrow.gif";
```

StationaryMargins

Sets the margins to be stationary. This property is read-only and should not be changed on the client.

0 = No

1 = Header

2 = Footer

3 = HeaderAndFooter

UniqueID

Gets a value that corresponds to the name of the element on the server. For a simple UltraWebGrid element hosted on an ASP.NET page, this will evaluate to the name of the element that was assigned in the development

environment (i.e. UltraWebGrid1.) For an UltraWebGrid element that is part of a user control, this will evaluate to the name of the user control concatenated with the name of the UltraWebGrid element. The names of the user and the UltraWebGrid element are separated with a colon in server-side code, but the colon is omitted on the client-side to make this property JavaScript friendly. So if you have a user control containing an UltraWebGrid, its unique ID on the server might be "MyWebControl:UltraWebGrid1" while the value returned by the **UniqueID** property on the client side would be "MyWebControlUltraWebGrid1". You can use this property to ensure that your client-side code references the correct server-based element.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
var thisGridId = grid.UniqueID;
```

ViewType

Gets an integer value that indicates the type of view that is displayed for the grid. This property is read-only and should not be changed on the client.

0 = Flat

1 = Heirarchical

2 = OutlookGroupBy

Methods

addSortColumn

Adds the specified column to the sorted columns array.

Parameters:

colID- The ID of the column to use when sorting.

clear- Boolean parameter that specifies whether the current contents of the sorted columns array should be cleared when the new column is added. If set to **true** the array is cleared and the specified column becomes the only column in the array and the only sort criterion. If set to **false**, the specified column is simply added to any existing columns in the array as an additional sort criterion.

alignGrid

Aligns the grid's activation rectangle to the upper-left corner of the data area. If a row is currently active, that row becomes the topmost row. If a cell is active, that cell's row is scrolled to the topmost position, and it's column is scrolled to the leftmost position.

Parameters:

param1- The_first_parameter.

param2- The_second_parameter.

beginEditTemplate

Opens the template for the currently active row. The template provides an alternate means for the user to enter row data using distinct elements.

Parameters:

None.

deleteSelectedRows

Deletes any rows that are currently selected in the grid.

Parameters:

None.

endEditTemplate

Closes the editing template for the currently active row. Changes to the template's controls can be either applied to row data or discarded.

Parameters:

saveChanges- A Boolean value that specifies whether changes made to the controls in the template should be applied to the data in the row. If set to **true**, changes will be applied. If set to **false**, changes will be discarded.

find

Searches through the grid data for a specified string. [Regular expression syntax](#) is supported for advanced searches.

Parameters:

regExp- The regular expression used to search the data. This may be omitted on subsequent calls to the **find** method; the previously specified expression will be used to perform the search.

searchUp- A Boolean value that specifies the direction of the search. If set to **true** the search is performed from bottom to top. If set to **false** (or omitted) the search is performed from top to bottom.

findNext

Searches through the grid data for a specified string, starting at the location of the previous find. [Regular expression syntax](#) is supported for advanced searches.

Parameters:

regExp- The regular expression used to search the data. This may be omitted on subsequent calls to the **find** method; the previously specified expression will be used to perform the search.

searchUp- A Boolean value that specifies the direction of the search. If set to **true** the search is performed from the position of the previous find to the top of the data. If set to **false** (or omitted) the search is performed from the position of the previous find to the bottom of the data.

getActiveCell

Returns a Cell object representing the active cell within the grid.

Parameters:

None.

getActiveRow

Returns a Row object representing the active row within the grid. The active row is the row that contains the active cell in the grid.

Parameters:

None.

selectCellRegion

Selects multiple cells in a rectangular region.

Parameters:

ce111- The beginning of the selection. This cell will serve as the anchor of the selection area, determining where the selection starts.

ce112- The end of the selection. This cell will be the last cell selected and will determine where the selection area stops.

selectColRegion

Selects multiple columns in the band.

Parameters:

co11- The beginning of the selection. This column will serve as the anchor of the selection range, determining where the selection starts.

co12- The end of the selection. This column will be the last column selected and will determine where the selection range stops.

selectRowRegion

Selects multiple rows in the band.

Parameters:

row1- The beginning of the selection. This row will serve as the anchor of the selection range, determining where the selection starts.

row2- The end of the selection. This row will be the last row selected and will determine where the selection range stops.

setActiveCell

Makes the specified cell element the active cell in the grid. This cell will have a border rectangle drawn around its perimeter.

Parameters:

cell- The Cell object to be made active.

setActiveRow

Makes the specified row element the active row in the grid. This row will have a border rectangle drawn around its perimeter.

Parameters:

row- The Row object to be made active.

sort

This method sorts the grid on the client-side, using the columns specified in the sorted columns array.

Parameters:

None.

sortColumn

Sorts the grid on the client-side using the specified column's contents as the sort criteria.

Parameters:

colID- The ID of the column to use when sorting.

shiftKey- Boolean parameter that specifies whether to perform a multi-column sort. If **true**, the specified column is added to the sorted columns array, and the sort is done based on the columns in the array. If **false** or omitted, only a single column is used. The contents of the sorted columns array are cleared and the specified column becomes the only one in the array.

unloadGrid

Destroys any allocated variables and arrays that have been created on the client-side by the control. This method should ONLY be invoked from the onUnload event of the page.

Parameters:

None.

Events

The **Events** object contains over 25 members that are used to specify JavaScript function handlers that are called in response to a variety of events occurring on the client. The **Event** member is only populated with a valid JavaScript function if the function name is set from the **Events** object of the **DisplayLayout** object on the server.

All events are called with at least two parameters: The grid name, and the ID of the element on which the event occurred. In the case of the **BeforeCellUpdate** event, an additional parameter is added which specifies the new value for the cell.

Note that **gn** is the parameter in the grid that the event came from, and **id** is the object that the event is on.

The members of the **Events** object are:

- AfterCellUpdate(gn, id)
- AfterColumnMove(gn, id)
- AfterColumnSizeChange(gn, id, width)
- AfterEnterEditMode(gn, id)
- AfterExitEditMode(gn, id)
- AfterRowActivate(gn, id)
- AfterRowCollapsed(gn, id)
- AfterRowDeleted(gn, id)
- AfterRowExpanded(gn, id)
- AfterRowInsert(gn, id)
- AfterRowSizeChange(gn, id, height)
- AfterRowTemplateClose(gn, id)
- AfterRowTemplateOpen(gn, id)
- AfterSelectChange(gn, id)
- AfterSortColumn(gn, id)
- BeforeCellChange(gn, id)

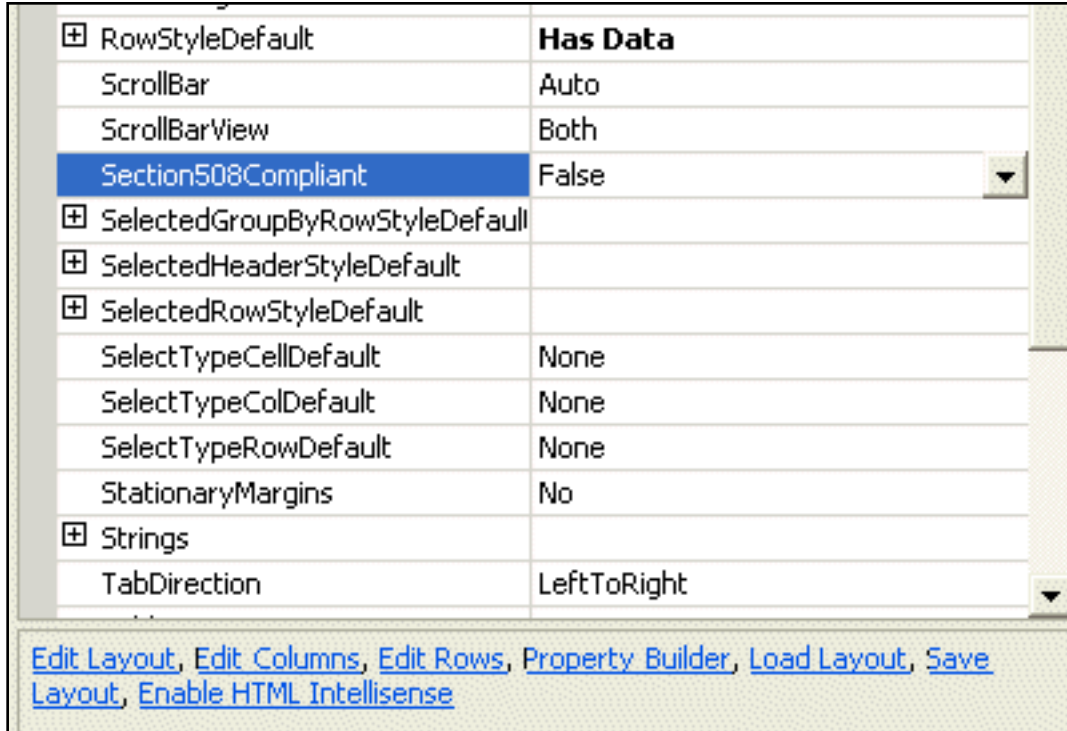
- BeforeCellUpdate(gn, id, newValue)
- BeforeColumnMove(gn, id)
- BeforeColumnSizeChange(gn, id, width)
- BeforeEnterEditMode(gn, id)
- BeforeExitEditMode(gn, id)
- BeforeRowActivate(gn, id)
- BeforeRowCollapsed(gn, id)
- BeforeRowDeleted(gn, id)
- BeforeRowExpanded(gn, id)
- BeforeRowInsert(gn, id)
- BeforeRowSizeChange(gn, id, height)
- BeforeRowTemplateClose(gn, id)
- BeforeRowTemplateOpen(gn, id)
- BeforeSelectChange(gn, id)
- BeforeSortColumn(gn, id)
- CellChange(gn, id)

- ClickCellButton(gn, id)
- ColumnDrag(gn, colId, insertBeforeColumnId)
- DbClick(gn, id)
- EditKeyDown(gn, id, keyCode)
- EditKeyUp(gn, id, keyCode)
- InitializeLayout(gn, id)
- InitializeRow(gn, id)
- KeyDown(gn, id, keyCode)
- KeyUp(gn, id, keyCode)
- MouseDown(gn, id, button)
- MouseOver(gn, id, type)
- MouseOut(gn, id, type)
- MouseUp(gn, id, button)
- TemplateUpdateCells(gn, templateId, cellId)
- TemplateUpdateControls(gn, templateId, cellId, value)
- ValueListSelChange(gn, valueListId, cellId)

Section 508 Compliance

Section 508 Compliance

NetAdvantage elements provide the ability to render **Section 508 Compliant** HTML to your browser. The **Section508Compliant** property under the **DisplayLayout** node of the **Section 508 Compliant** elements is set to **False** by default. Setting this property to **True** enables the **Section 508 Compliant** functionality.



Section 508 Compliance involves making sure that your application can be used effectively by end-users with disabilities.

This primarily means that you must ensure some level of accessibility to:

- Users with low vision or no vision who may be using assistive technologies such as screen magnifier or screen reader software.
- Users with limited movement who may be using alternative input methods instead of the standard keyboard and mouse combination. This might include simply a keyboard with no mouse, or some kind of alternate pointing device that does not have the precision of a mouse.

(**Section 508** compliance also includes considerations for users with hearing disabilities. However, audio considerations are outside the scope of this documentation, because **Infragistics** products primarily provide visual interfaces.)

If you are developing applications for a United States Federal agency, accessibility is a requirement of your application under **Section 508** of the Rehabilitation Act of 1973. For more information, you can refer to the governmental web site <http://www.section508.gov/>. Keep in mind that Infragistics only provides the tools for you to use. It is ultimately the responsibility of the developer to design an interface that is usable by people with disabilities.

Introduction - Setting CopyLocal To True

In order to run applications that use the Infragistics Web Controls on your machine, you will need to set the Copy Local flag to True for the `Infragistics.WebUI.Shared.dll` Assembly. This is because that assembly is registered into the Global Assembly Cache by the Windows Forms products, but Web Applications, by default, do not look in the GAC to resolve assembly references at runtime. Therefore, the `Infragistics.WebUI.Shared` assembly must be copied to the local project directory along with the respective Web control's assembly in order to run the Web application under IIS.

Deploying Your ASP.NET Application

For Web Forms, you will generally deploy the application to a web server that hosts the application for clients to access through their web browser.

To deploy a web application using Infragistics UltraWebGrid, take these steps:

1. Copy the contents of the application directory and `bin\` directory from your development machine to the corresponding location on the deployment server.
2. The application must be created using IIS on the web server so that it can be accessed from client machines.
3. Copy the image and JavaScript files to the appropriate directory on the server, as indicated below.
4. Optionally modify the properties of the controls that indicate the location of the JavaScript and image files. This is only necessary if you are required to use virtual paths on the web server other than those outlined in the next section. See [Working in a Hosted Environment](#) for more details.

Script and Image Files

UltraWebGrid installs images and JavaScript files into a standard directory structure. The location for scripts and images is determined by reading the registry entry: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\INetStp\PathWWWRoot`. If that is set to the IIS default then these files are installed to `C:\inetpub\wwwroot\aspnet_client\infragistics\WebGrid3`. If this registry entry is not found then the files are installed to `C:\Program Files\Common Files\Infragistics\Web\WebGrid3`. The virtual directories that map to these physical directories are named `ig_common\webgrid3`.

UltraWebGrid has a Client-Side Object Model (CSOM). To support this functionality, a common JavaScript file is shared by all of the Infragistics Web controls. The location of this file is specified using the `JavaScriptFileNameCommon` property. By default, it is in a virtual directory `/ig_common/scripts/ig_csom.js` which maps to the physical path `c:\inetpub\wwwroot\aspnet_client\Infragistics\Scripts\ig_csom.js`.

Installation of JavaScript Files

Correct installation of the JavaScript file requires that it be copied to the `/ig_common/WebGrid3` virtual directory within IIS in order to be accessed at runtime. By default, the product installation maps this directory to `C:\inetpub\wwwroot\aspnet_client\Infragistics\WebGrid3`. If you receive updates for JavaScript files, they should be placed into this directory to overwrite the older copies.

Customizing Installation Parameters

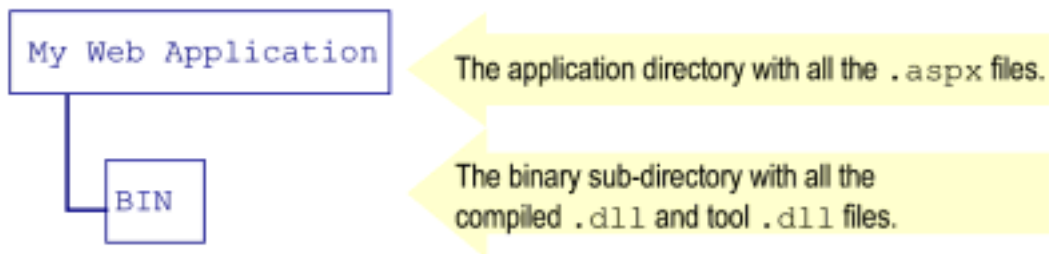
Infragistics Web controls have properties that allow the application to control the location of script files and image files. There are `ImageDirectory`, `JavaScriptFilename` and `JavaScriptFileNameCommon` properties available which are given default values that "just work" but which can also be customized for any deployment scenario.

Working in a Hosted Environment

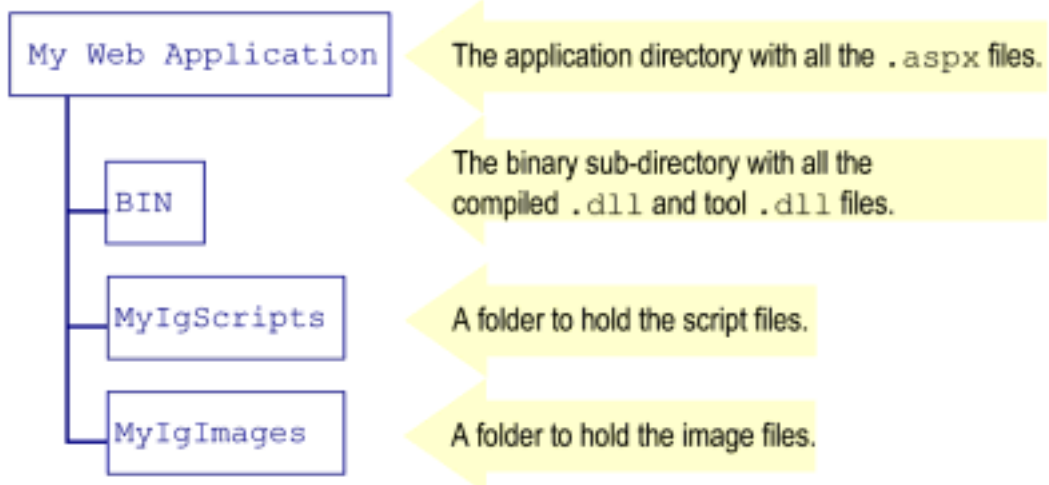
When installing an ASP.NET application onto a hosted server, often the developer will not be able to set up virtual directories on the hosting machine. In these cases, relative pathing can be used so that the project can be deployed on a hosted server.

In order to accomplish this, the JavaScript path and the image path must be changed to a valid relative path and the JavaScript files and image files moved into the proper relative path.

For example, if a web application called `MyWebApplication` were to be deployed, the normal structure of the deployment for the project would be:



The JavaScript files and image files would be located in their virtual directories and the application would be able to run. To convert this for a web-hosting situation one could set their application up like this:



Inside the application, on the various Infragistics Web tools, you would change the **JavaScriptFileName** and **ImageDirectory** properties to reflect the relative path. So in the case of WebGrid the default values would be:

```
JavaScriptFileName= "/ig_common/WebGrid3/ig_WebGrid.js"
ImageDirectory= "/ig_common/WebGrid3/"
JavaScriptFileNameCommon= "/ig_common/scripts/ig_csom.js"
```

To change this for MyWebApplication one would have to change the values to the following:

```
JavaScriptFileName= "MyIgScripts/ig_WebGrid.js"
ImageDirectory= "MyIgImages/"
JavaScriptFileNameCommon= "MyIgScripts/ig_csom.js"
```

After this change, you would only have to move the JavaScript files and image files from your development machine into the newly created folders in the web-hosting environment.

Distributable Files

When deploying a .NET application, you must distribute certain files in addition to your executable. If your application contains one or more Infragistics controls, you will have to distribute or deploy one or more Infragistics assembly files as part of your application. This topic outlines the files that you must re-distribute.

Note Any files not specifically covered by this topic should be considered *non-redistributable*. Files that are included with your Infragistics product but not listed here are not licensed for distribution, and should not be copied to, moved to or shared with any machine other than the one on which the licensed Infragistics product is installed.

In order to deploy your application, you will have to re-distribute the following files. These files are required for any application that makes use of the UltraWinExplorerBar:

File Name	Description
Infragistics.WebUI.UltraWebGrid.v3.dll	.NET Assembly containing the UltraWebGrid control.
Infragistics.WebUI.UltraWebCombo.v3.dll	.NET Assembly containing the UltraWebCombo control.
Infragistics.WebUI.UltraWebGrid.ExcelExport.v3.dll	.NET Assembly containing the UltraWebGrid Excel Export functionality.

In addition to the above assembly files, the deployment of any Infragistics Web-based User Interface (ASP.NET) product also requires the re-distribution of the following files:

File Name	Description
Infragistics.WebUI.Shared.v2.dll	Infragistics Web Application Shared support assembly for .NET

The following files are required to support this product in the ASP.NET environment and must be properly deployed on any server that will be hosting the ASP.NET application containing the control(s).

Note Installing the product on a development machine deploys these elements to their correct locations based on the development machine's local web server. The files in these locations will work with the control's default settings. You can mirror the development machine's configuration to the production server, and the controls will then work with their default settings on the production server. If the configuration of the production server will differ from that of the development machine, you will need to make corresponding changes to the default values of the Infragistics Web controls in your application.

The following files are required to support the product in an ASP.NET application:

File Name	Description
ig_webcombo.js	JavaScript Support File for the UltraWebCombo control.
ig_WebGrid.js ig_WebGrid_an.js ig_WebGrid_dl.js ig_WebGrid_dom.js ig_WebGrid_ft.js ig_WebGrid_gb.js ig_WebGrid_ie.js ig_WebGrid_ie6.js ig_WebGrid_kb.js ig_WebGrid_ml.js ig_WebGrid_nn.js ig_WebGrid_ro.js ig_WebGrid_srt.js	JavaScript support files for the UltraWebGrid control.
(multiple images)	Various images required to support the control are installed on your development machine, and must be deployed to the server in order for the control to function. For the location of these image files and a description of how to deploy them, see the Deployment topic.

The location of these files, as well as full details on deploying your application are provided in the [Deployment](#) topic.

Introduction - ViewState

ViewState is the mechanism that allows program state to be maintained in the disconnected and stateless world of HTTP. What this means is that changes made to the grid under program control can be carried forward through multiple page post-backs and recreations of the grid on the server by storing important state information in ViewState.

Because there are numerous objects and properties of UltraWebGrid along with the potential for many rows of data, view state considerations can become quite important in development decisions with UltraWebGrid.

Some of the factors that go into the decision making process concerning viewstate are:

1. How much data, in total, are you concerned with? How many bands, columns and rows are potentially going to be retrieved and displayed in the grid? The more data involved, the larger and more expensive viewstate overhead becomes.
2. How expensive is the cost of reconnecting to the data on each post-back? If the grid can be reloaded and reconfigured efficiently, viewstate is not as important. This is what most ASP and other server-side development practice has been in the past.
3. How frequently will post-backs be necessary? The more work that can be done on the client without server post-backs, the less viewstate overhead becomes an issue.
4. Are data sorting or Outlook GroupBy features involved? Sorting and Outlook GroupBy functionality require server post-backs.
5. What is the minimum bandwidth of the connection to the server? Are there dial-up connections involved or is it an intranet application?
6. What are the scalability requirements? Can the viewstate be kept on the server in session state?

Guidelines for viewstate handling options

EnableViewState = False.

With this option there will be no viewstate at all for the control. While this option has the lowest overhead in bandwidth, it also has significant limitations. If the page needs to be round tripped to the server, the data must be rebound on each post-back. Any information about the grid that changes during the session will be lost unless it is saved to the data store or re-applied on each post-back. If paging is in effect and database updating is required, the updates must be applied to the current page before the new page is bound.

Having no viewstate at all is a good option if there are many rows of data on a single page, or the data is read-only and there are only limited requirements for event handling and post-backs.

EnableViewState = True, EnableInternalRowsManagement = False

With this option there is viewstate for the current page of the grid but no more. This means that the current page can be round-tripped as much as necessary without the need to re-bind to data, and any changes made to the grid will be maintained. If paging or sorting is involved, the grid will need to bind to data once again and all information from other pages will be lost. This option is usually the best in terms of trade-off between the convenience of viewstate and the overhead of moving it between the server and client. Another option with these settings is to keep the viewstate on the server using session state.

EnableViewState = True, EnableInternalRowsManagement = True

In this case the grid will read all rows from the data source and store them in viewstate while only rendering the current page. The data source does not need to be reloaded on any post-backs unless the database has been updated. The grid can perform all paging, sorting and GroupBy functionality using the rows that have been saved to viewstate. This option may create very large viewstate overhead if a data source is large. However, it can be good option for medium and small data requirements or if viewstate is maintained in session state on the server.

Keeping ViewState in SessionState

If the ViewState requirements for a page are large, and large bandwidth is not available or guaranteed, then placing the ViewState data into SessionState is an option.

To put UltraWebGrid ViewState into SessionState, add the following code to the Page object:

In Visual Basic:

```
Protected Overrides Function LoadPageStateFromPersistenceMedium() As Object
    Return Me.Session("GridState")
End Function

Protected Overrides Sub SavePageStateToPersistenceMedium(ByVal viewState As Object)
    Me.Session.Add("GridState", viewState)
End Sub
```

In C#:

```
protected override object LoadPageStateFromPersistenceMedium()
{
    object state = this.Session["GridState"];
    return state;
}

protected override void SavePageStateToPersistenceMedium(object viewState)
{
    this.Session.Add("GridState", viewState);
}
```


Tabular Display and the ViewType Property

The **ViewType** property determines the structure and capability of the current UltraWebGrid. There are three options for ViewType: flat, hierarchical and OutlookGroupBy.

The basic unit of display for the grid is the HTML table consisting of Columns, Rows, and Cells. UltraWebGrid is optimized to display this information quickly and efficiently, and it provides a straightforward programming interface for working with flat, tabular data. The top-level UltraWebGrid object contains the Columns collection property which allows the top-level columns of the grid to be configured.

Also on the top-level, the Rows collection supplies direct access to the rows of information in the grid. Within each Row object is a Cells collection that is used to store the individual values of the table cells. All properties concerning the appearance and behavior of the grid can be accessed using the DisplayLayout object which is described below.

To indicate that the grid is to display flat tabular data, use the following setting:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.Flat
```

In C#:

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.Flat;
```

Hierarchical representations of data can also be displayed with the UltraWebGrid. To ensure that child rows are visible, the **ViewType** property must be set to Hierarchical.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.Hierarchical
```

In C#:

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.Hierarchical;
```

Another feature of UltraWebGrid is the ability to view rows in GroupBy mode consistent with the GroupBy grouping in Microsoft Outlook. The following code would set the **ViewType** property allowing for grouping of rows. Remember, the **AllowGroupBy** property must be set as well.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.OutlookGroupBy  
UltraWebGrid1.DisplayLayout.Bands(0).Columns(0).IsGroupByColumn = True
```

In C#:

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.OutlookGroupBy;  
UltraWebGrid1.DisplayLayout.Bands[0].Columns[0].IsGroupByColumn = True
```

Additional information on the GroupBy view type can be found within the [Introduction - Outlook GroupBy Mode](#) topic.

Introduction - Cross-Browser Compatibility

UltraWebGrid targets three different levels of capability depending on which browser is in effect on the client. The three different targets are:

1. Internet Explorer version 5.5 and above
2. Netscape 6.0 and above
3. Down-level. Earlier versions of the above browsers and all other browsers.

Targets 1 and 2 above are considered up-level browsers and the third target indicates down-level browser support.

Because of the popularity and ubiquitous use of Microsoft's Internet Explorer versions 5.5 and above, UltraWebGrid is optimized and most fully functional in that environment. While all server-side functionality is also provided for the other targets, it is modified to some degree in order not to exceed the capabilities of those browsers.

The functional differences between the three targets are outlined below:

Target 2, Netscape 6.0 provides all functionality of Target 1 with the following exceptions:

- Outlook GroupBy is supported by clicking on images in column headers rather than by drag and drop.
- The ActiveRow and ActiveCell rectangle is not displayed.

Target 3, down-level support differs from target 1 in the following ways:

- There is no JavaScript executed on the client. So client-side events and the client-side API are not available.
- All behavior is implemented through server-side post-backs. This includes cell, row and column selection, cell editing, row deleting and row adding.
- Column and row resizing is not supported.
- Outlook GroupBy is supported by clicking on images in column headers rather than by drag and drop.

Introduction - DisplayLayout

The DisplayLayout object is the central location for the properties and data collections of UltraWebGrid. When programming with the grid, it is very common to use *UltraWebGrid.DisplayLayout.SomeProperty* syntax to change the look or behavior of the control. The DisplayLayout contains numerous default values for styles and behaviors throughout the grid. DisplayLayout is also the default property for the grid at design time.

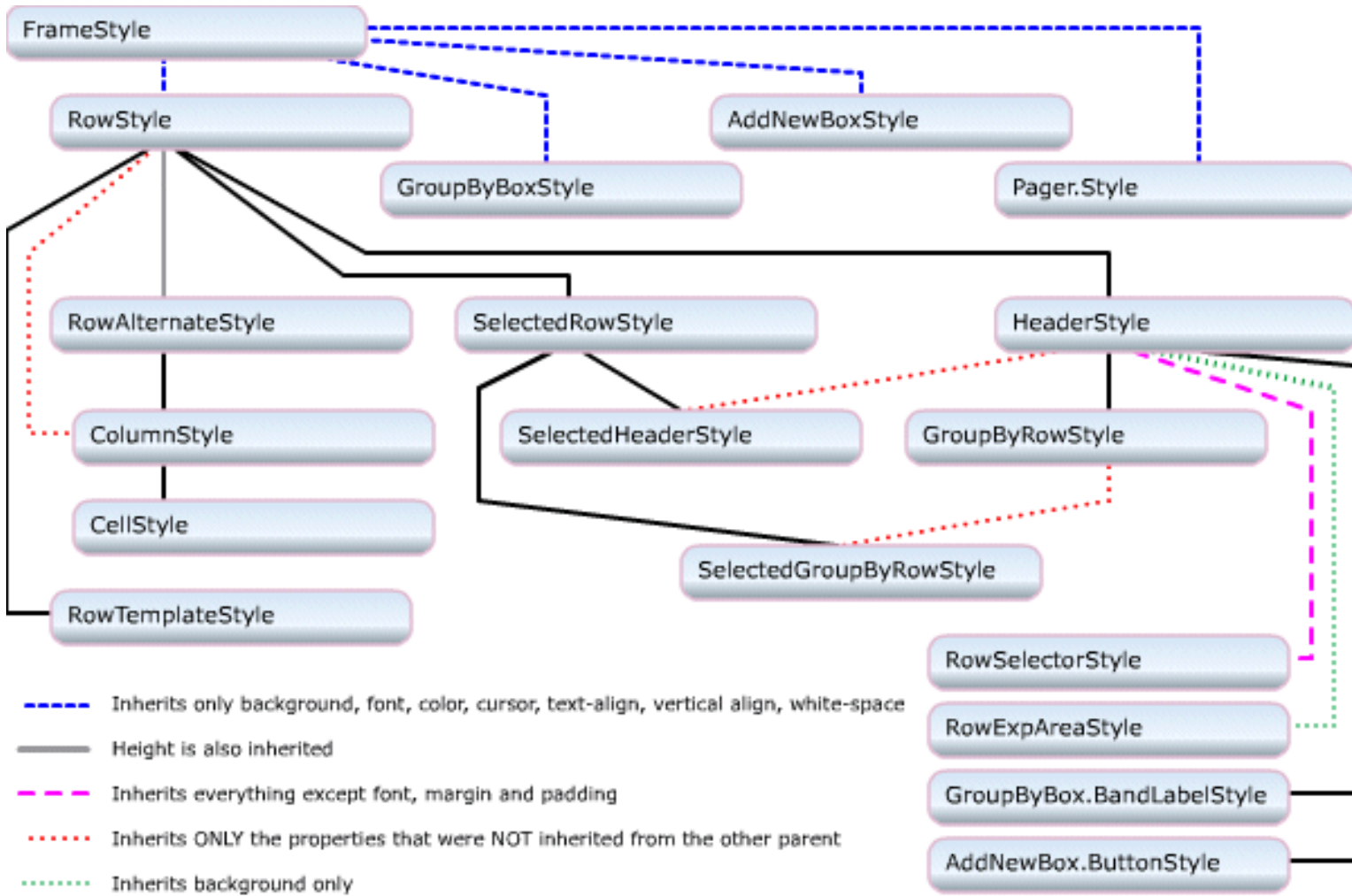
If the ViewStyle of the grid is Flat, then the DisplayLayout default values will be all that is needed to configure the styles and behaviors of the grid. If the grid is Hierarchical, then additional control and customization can be achieved by setting default values that apply to one or more bands, and then overriding those values by using individual setting for other bands. This allows the information at each level of the band hierarchy to have an associated look and behavior with a minimum of programming effort.

Style Inheritance

The DisplayLayout object's **MergeStyles** property controls the way the styles are merged in the grid. By default it is set to True, which means that styles inherit properties from parent objects based on the inheritance hierarchy. With this setting, the developer does not have to specify the appearance settings for every object in the control, or manually set every property on an object when they want to change only single property's value.

All properties that can inherit their values have a "use default" setting. If a property is set to "use default" it will inherit its value from the parent object. Most appearance-related properties start out set to their "use default" values.

The following diagram illustrates how styles are inherited among the objects of the UltraWebGrid:



If for some reason the developer does not want to inherit style properties, **MergeStyles** can be set to False. In this case, all styles will must be supplied with the desired property values.

Introduction - Client-Side Object Model

In addition to firing client-side events, UltraWebGrid also supports a robust object model on the client. Grid, Bands, Columns, Rows and Cells can be accessed and modified substantially, though not totally, without the need for server-side post-backs. While client-side programming is powerful and convenient, not every aspect of functionality can be supported to the degree that it can on the server. Also, the client-side object model is not available in down-level browser support.

The UltraWebGrid Client-Side Object Model has [its own section](#) in the help file. [Go there](#) to find out more.

Introduction - Events

UltraWebGrid exposes numerous events that can be handled either on the client, using JavaScript, or on the server using .NET languages. For client-side events, the DisplayLayout contains a **ClientSideEvents** object where JavaScript function names can be specified for those events of interest. If specified, the function will be called on the client whenever the event occurs. The event handler on the client can cancel the event by returning non-zero from the handler, and it can cancel any possible post-back for server-side handling by calling the utility function `igtbl_cancelPostBack()`.

On the server, there are events fired for events occurring on the server as well as for events occurring on the client. For instance, **DataBinding**, **InitializeLayout**, **InitializeBand** and **InitializeRow** are fired in sequence during the data binding process on the server. **ActiveCellChange**, **ActiveRowChange**, **CellButtonClick** and **DbClick** are examples of events that occur on the client but can be fired on the server. Events are fired from the client to the server only if the application has registered a handler for the event. There are no post-backs for client-side events otherwise.

If handlers are registered, most events from the client are fired to the server as they occur. This is the case for the **AddRow**, **DeleteRow** and **UpdateCell** events. The **AddRow** event will fire as soon as a row is added to the grid. The **DeleteRow** event will fire as soon as the delete key is pressed and the selected rows are removed. The **UpdateCell** event will fire as soon as editing ends on a cell. Three similar events do not fire immediately however: **AddRowBatch**, **DeleteRowBatch**, and **UpdateCellBatch** fire on the server only after a normal post-back is performed from the client. These three events can be used to group multiple updates for the grid into a single post-back.

Introduction Data Binding

UltraWebGrid supports binding to data sources of various kinds. The most common is a database table or a dataset containing multiple tables with relations defined between them. At the most basic level, UltraWebGrid can bind to any object that supports the generic `IEnumerable` interface. Using reflection, the objects in the enumeration are read and added to a Band's Columns collection according to the property names of the object type being enumerated. Each property becomes a column. Any columns that are not to be rendered and visible need to have their **Hidden** property set to True.

If a column in the enumerated object is of type `IEnumerable`, UltraWebGrid will drill down on it, create a child band, and populate the rows at that level in the same manner, using the properties of the child enumeration. Only one child band is permitted per band. Therefore, if there are multiple enumerable properties on an object, UltraWebGrid will read the first one it encounters unless told otherwise in response to the `InitializeBand` event. In that event, the application can set the `ChildBandColumnName` to identify which property name or relation name to enumerate. If "NoChildBands" is specified, no child bands will be enumerated.

There are several events relevant to the data binding process that are listed here in the order they are fired:

1. *DataBinding* - This event is called as soon as the `DataBind` method is called. Since `DataBind` may be called from many different points in the application, its purpose is to allow for a single point in code to set up, configure and assign the `DataSource/DataMember` properties of the grid.
2. *InitializeBand* - This event can be used to configure individual bands of a multi band layout. It is also the place where the `ChildBandColumnName` property of the band can be set to specify which enumerated column of the band should be used to populate a child band.
3. *InitializeLayout* - This event is fired when all datasource metadata information has been read in. It can be used to configure the grid, bands, or columns with styles and behavior properties. It is also the point at which additional unbound columns may be added to the column structure.
4. *InitializeRow* - This event is fired as each row is added to the rows collection. It provides an opportunity to alter the contents of cells as well as to populate cells in unbound columns. It can also be used to set styles on individual cells.
5. *InitializeFooter* - This event is fired when footers are in effect for a band and the end of a data island has been reached. It provides an opportunity to configure the contents of the footer by setting the text and values of cells within the footer row.

These events do not fire at all unless the grid is bound to a datasource using the `DataBind` method.

Exporting Grid Data to Excel Format

This topic gives a general overview of the Excel exporting process and the elements involved. For step-by-step instructions on how to export Excel data, see the [Exporting Grid Data To Excel](#) topic in the Task-Based Help section of the help.

With this release, the UltraWebGrid gains the ability to export data in the native Microsoft Excel spreadsheet format. Exporting to Excel format is a process similar to rendering grid data, in that a developer can control how Layouts and Appearances are applied to the data before export, and which data is included or excluded. Excel export takes advantage of many of the advanced formatting features of Excel, and export is not limited to grid data; column headers, footer rows, and more can all be included in the export.

Excel exporting is handled by two additional assemblies. The Infragistics.WebUI.UltraWebGrid.ExcelExport assembly interacts with the UltraWebGrid in managing all collaboration that takes place during export. (This assembly has a dependency on the UltraWebGrid assembly.) The Infragistics.Excel assembly communicates only with the Infragistics.WebUI.UltraWebGrid.ExcelExport assembly and handles the nuts-and-bolts details of exporting data into the native Excel file format. The **UltraWebGridExcelExporter** control must be added to the web application in order to export cell data from an **UltraWebGrid**. Adding the control automatically adds a reference to the Infragistics.Excel assembly into the application's project.

The export process begins by invoking the **Export** method of the **UltraWebGridExcelExporter**, passing it the **UltraWebGrid** from which to export grid data, and optionally the filename of the Excel file you wish to create. Developers have the opportunity to specify several customizable aspects of the export either through calling an appropriate overload of the **Export** method, or declaratively setting properties on the **UltraWebGridExcelExporter** control.

Next the **UltraWebGridExcelExporter** walks the **UltraWebGrid** during the export process. Communication is conducted by the *IUltraWebGridExporter* interface. The grid exposes this interface, which is used to export data in a generic fashion. The grid calls methods of this interface to export grid objects such as rows, headers, footers, and so on. The **UltraWebGridExcelExporter** implements these methods to convert the data coming from the grid into corresponding "objects" in the native Excel format. (Note that these may not be objects in the strict sense of the word, they may simply be expressed as text and formatting attributes that will be applied to one or more Excel cells.)

As the ExcelExporter is processing the data from the grid, it raises events allowing the developer to have control over the export process. Developers may perform various tasks in these events, affecting both the selection of what data should be included in the export and the disposition of that data in the resulting Excel spreadsheet. Events include **BeginExport**, **InitializeRow**, **InitializeColumn**, **CellExporting**, **CellExported**, **RowExporting**, **RowExported** and **EndExport**. This is only a partial listing, as there are multiple events for the different types of objects supported by the export. For a complete list, see [UltraWebGridExcelExporter Class Members](#).

Internally, the **UltraWebGridExcelExporter** passes the **UltraWebGrid** a helper class that implements the *IUltraWebGridExporter* interface. The grid walks its data from top-to-bottom, and left-to-right, invoking the methods of the passed-in helper class' interface to export different types of grid objects. The implementations of those methods in the helper class call corresponding methods in **UltraGridExcelExporter**, which in turn raises events as it converts the grid data into Excel data. Developers may handle these events to customize this conversion process, or implement a conversion that is not possible to perform automatically. The export continues until the entire grid is exported, or the process is terminated. Finally, the Excel data is written into a file or stream that is transmitted to the end user's browser.

While exposed as a dynamically-configurable control developers may place on the web form, the export functionality of the **UltraWebGridExcelExporter** has greater similarity to an ISAPI filter in that it creates a stream in the HTTP response that is downloaded by the end user. In doing so, no other controls on the web form are permitted to render HTML content into the same response stream as the exported Excel spreadsheet.

Introduction - LoadOnDemand

The LoadOnDemand feature provides a way for you to prevent the unexpanded rows in a hierarchical grid from being sent down to the browser, thus saving on page transmission time.

Automatic LoadOnDemand is designed to be useful in certain situations, but is not appropriate for every situation. It is most useful when the number of top level rows is small, but there may be a large number of rows in Band 1 and higher. The feature is designed to minimize the total number of rows sent over the wire to the browser at one time. The trade off (and yes it is a trade off) is that data must be bound to the grid on each post-back. It is also advisable to have [EnableInternalRowsManagement](#) set to false and possibly ViewState turned off as well in order to gain any advantage from the LoadOnDemand feature. Otherwise, many of the rows that are not needed will travel across the wire via the ViewState.

Manual LoadOnDemand is useful when child rows for a particular row can be manually bound in application code. The [DemandLoad](#) event is fired to the application when the user has clicked on a row expansion button that has not yet been loaded with its children. The application must then be able to fetch the records that belong to the expanded row and add them to the Rows collection of that row. Using LoadOnDemand.Manual, it is necessary to have Band objects created for each level of Rows that may be expanded using plus (+) images. If there is no Band object for rows that may be expanded, the WebGrid will not display a plus sign next to rows in the parent Band.

Introduction - Outlook GroupBy Mode

The Outlook GroupBy interface allows users to select various columns on which the row data will be sorted in either ascending or descending order. Once sorted, the rows with common values are grouped together under a single row that displays the common value of its children. UltraWebGrid creates an additional row for each common value in the sort order and adds it to the row hierarchy in order to carry out the grouping function. A row expansion indicator is placed next to the GroupBy row to indicate that the row can be expanded or collapsed for easier viewing.

Outlook GroupBy is enabled on the server using the following code:

```
UltraWebGrid1.DisplayLayout.ViewType = ViewType.OutlookGroupBy
```

Individual columns can then be permitted or not permitted to be grouped by setting the Column's **AllowGroupBy** property to AllowGroupBy.Yes or AllowGroupBy.No. This setting controls whether the column can be grouped by the user on the client.

Grouped columns can be specified in code by setting the Column's **IsGroupByColumn** property to True. To ungroup a column, set its IsGroupByColumn property to False.

Note The grouping of rows according to the GroupBy column configuration and the sorting of the row data is performed as the last step of the DataBind operation. During this procedure, all existing GroupBy rows are destroyed and new ones are created that conform to the GroupBy conditions in effect. If additional GroupBy changes are made after this point, such as adding or removing a GroupBy column in code, UltraWebGrid will rebuild the GroupBy row hierarchy again during the PreRender event. Therefore, any customized GroupBy row style information will need to be set from the application by responding to the PreRender event so that it occurs after the final GroupBy rows have been created. Normal row styles are not affected.

Introduction - Bands

As mentioned under Hierarchical Display, the Band object controls style and behavior properties for all rows at a particular level of the grid hierarchy. In addition to the Columns collection, which describes the data and contents of each cell of each row, the Band object contains many properties that can override the default values specified in the DisplayLayout object. For instance, the default for AllowUpdates may be false for the grid overall, but `Band(1).AllowUpdate` can be set to true to allow updating information in the rows of that band only.

There is always at least one Band object for the grid and it is referred to as `Band(0)`. That is the top-level band consisting of the rows that would appear if the grid was displaying flat data only. If there are no other Bands, then the DisplayLayout defaults are enough to control the look and feel of the flat, tabular display. If multiple bands are being used to display hierarchical data, the DisplayLayout object should be used to set the values common to most bands, and the individual Band objects should be used to set those properties that are unique to particular bands.

Introduction - Rows

You will notice that the Band object itself does not contain a Rows collection. The Rows collection is exposed by the main UltraWebGrid object only. This is because the rows are the raw data of the grid. In a manner of speaking, they get plugged into the layout. The DisplayLayout, Bands and Columns are there to shape and contain the data but they are separate from the data. Therefore, the data hierarchy of the grid is implicit in the tree structure of the Rows collection itself. Each Row of the toplevel Rows collection has a Rows collection of its own which, in turn, contains Row objects that may contain child rows. The Band information is then applied at the proper level points of the hierarchy as the row data is rendered. For example, the rows structure of a simple hierarchy might appear as follow:

```

Row
  Row
    Row
    Row
  Row
  Row
  Row
  Row
Row
  Row
  Row
Row

```

In this example there are three rows in Band(0), six rows in Band(1), and four rows in Band(2). However, none of the rows are actually owned by a Band, they are owned by the Rows collection of their parent Row object. The rows of Band(0) are exposed as the Rows collection of the UltraWebGrid object.

When Outlook GroupBy is in effect, (discussed above) the GroupBy rows containing the grouped, common data fall into the same tree hierarchy of parent and child rows as the normal hierarchical arrangement. This means that a row's position in the tree hierarchy can change depending on how many columns are grouped.

For example, the same row structure as before may become structured as follows after being sorted and grouped:

```

GroupByRow
  Row
    GroupByRow
      Row
        Row
        Row
        Row
      Row
      Row
    GroupByRow
      Row
      Row
  Row
  Row
  Row
Row

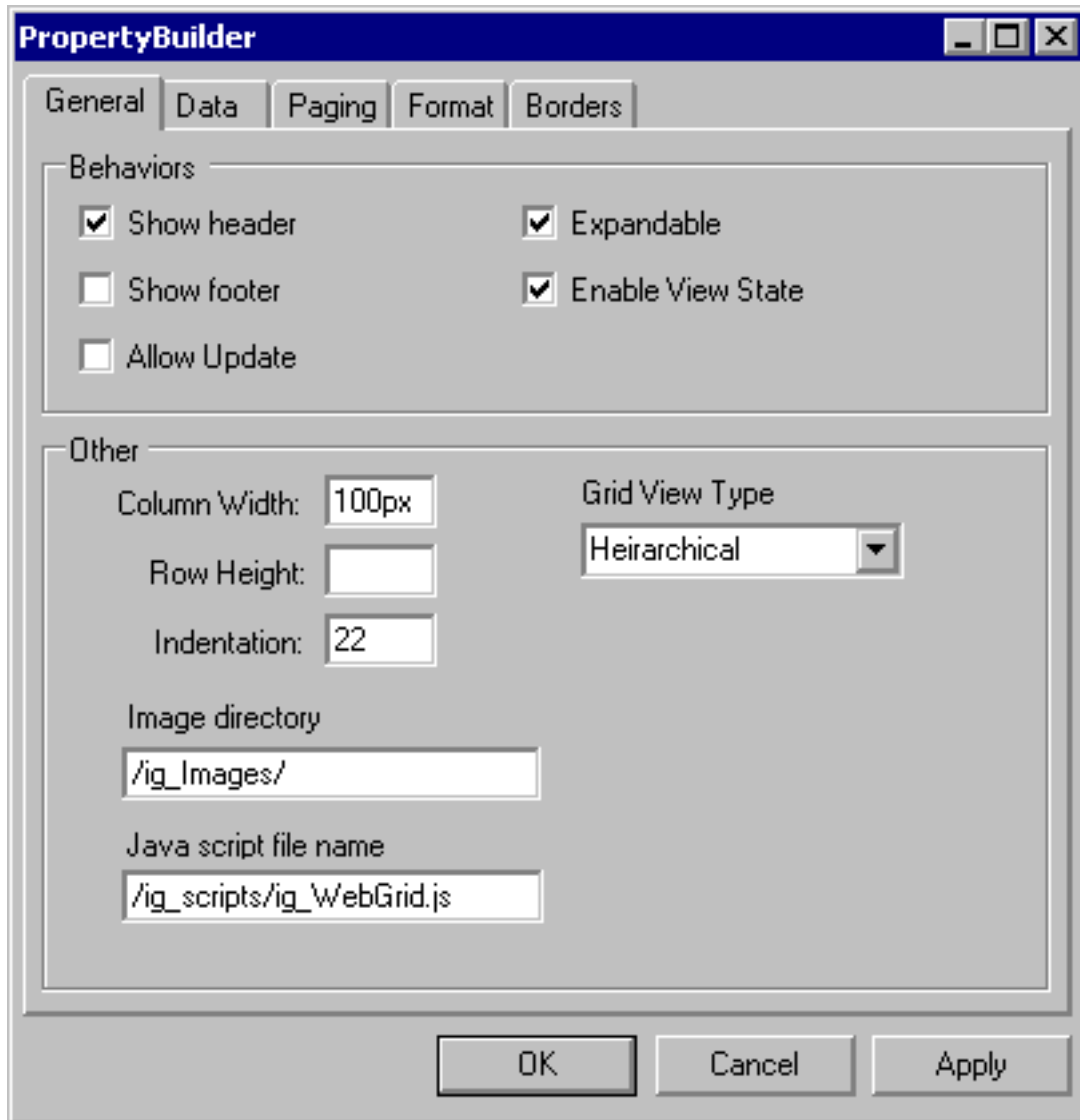
```

Here, GroupBy rows are added to the hierarchy to reflect the points where common cell values are located within the data. (Cell values are not shown here).

The Property Builder Dialog

The Property Builder dialog provides a centralized location where you can quickly set up a variety of the UltraWebGrid's properties using an interactive interface.

As you make changes, you can click the "Apply" button at any time to apply your changes from the dialog to the control. Click the "OK" button to apply any pending changes and close the dialog. The "Cancel" button closes the dialog without applying any pending changes. Clicking "Cancel" will not negate the effects of any changes that have already been applied.



General Tab

The first tab of the Property Builder dialog is the General tab. You use this section of the dialog to set properties that determine the overall look and behavior of the grid. To set up grid behavior, you can specify whether the grid should show headers and footers, whether updates to the data source will be allowed, whether grid rows can be expanded and whether [ViewState](#) information will be enabled for the client.

The **General** tab also gives you a place to set the default column width, row height and band indentation for the control. You can specify whether the grid will display with a flat, hierarchical or GroupBy style. You can specify the root directory on the server that will hold the images used to render the control, and the name of the JavaScript file that will provide the client-side functionality for the control.

Data Tab

The Data tab is used to specify how the client can interact with the data source. You can enable or disable the

addition of new records, the deletion of records, the updating of record data, and the sorting of grid data. Any behavior you choose will be enabled in the grid.

Note Enabling a data interaction feature using this tab will usually not be enough to actually implement the feature in your application. Typically, you will also have to set certain other properties and write code to handle parts of the client-side interaction. For more information, see the topics for the properties that correspond to the settings of this tab. ([AllowAddNewDefault](#), [AllowDeleteDefault](#), [AllowUpdateDefault](#) and [AllowSortingDefault](#).)

Paging Tab

Use the Paging tab to set up the paging mechanism the UltraWebGrid uses when record scrolling is either unavailable or purposely disabled. You can choose to allow paging (which disables scrolling) on this tab, then use the other settings to set up how paging will occur. Choose the number of rows that will appear per page. You can also control the placement and appearance of the elements of the paging interface; whether they appear at the top of the page, bottom of the page or both; how the paging links will be aligned, whether you will use a numeric or a previous/next interface, and the text that should appear on the "Previous" and "Next" paging buttons.

Format Tab

The Format tab provides a hierarchical interface you can use to quickly select the different types of items that make up the grid and apply formatting attributes to them. This tab is used for quick and basic formatting such as foreground and background colors, font face, size and attributes, and text alignment.

Borders Tab

The Borders tab is a simple interface to setting the attributes used by the grid for cell borders and margins. You can use this screen to set cell spacing and cell padding, the types of grid lines to display (horizontal, vertical or both) as well as the color and thickness of cell borders.

Client-Side Object Model - Introduction And Overview

While the WebGrid control is first and foremost a server based element, it also exposes a sophisticated, object oriented programming model on the client that developers can utilize to create more powerful, performant and flexible web applications. The Client Side Object Model, (CSOM), consists of JavaScript based objects, properties, methods and events that encapsulate much of the functionality and capabilities of the server side element, but without requiring costly postbacks for their implementation. While in some respects the CSOM provides more limited functionality and is not as well integrated into the Visual Studio environment as the server side element, the advantages of client-side functionality often outweigh these limitations.

The primary means through which a page developer interacts with the CSOM is by responding to client side events and altering the WebGrid object model in various ways to accomplish the required objectives. For instance, it may be required that, upon double clicking on a column header, the cell values of that column must be summed and the result placed into a different cell of the grid. This would be accomplished by entering a function name into the grid.DisplayLayout.ClientSideEvents.DbClick property on the server and writing a JavaScript function that would implement the required functionality when called in response to the mouse double-click event.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ClientSideEvents.DbClickHandler = "MyDbClick"
```

In C#:

```
UltraWebGrid1.DisplayLayout.ClientSideEvents.DbClickHandler = "MyDbClick";
```

The JavaScript function itself can be written directly into the aspx page as follows:

```
<script language='javascript'>
function MyDbClick(gridId, cellId) {
}
</script>
```

...or, it can be added to the Page object in the code behind file as follows:

In Visual Basic:

```
Dim jscript as String
jscript = "<script language='javascript'>function MyDbClick(gridId, cellId) { }</script>"
Page.RegisterClientScriptBlock("MyHandler", jscript)
```

In C#:

```
string jscript = "<script language='javascript'>function MyDbClick(gridId, cellId) { }</script>";
Page.RegisterClientScriptBlock("MyHandler", jscript);
```

The samples that ship with WebGrid generally place the JavaScript event handlers directly into the aspx page. Once your JavaScript function has been called on the client in response to an event occurring within WebGrid, the complete object model of the grid can be utilized. Accessing the objects can be accomplished in several ways depending on which objects are needed. The two primary points of entry into the CSOM are the main object, in this case the grid, and the target object of the event, which is most often a cell.

The gridId is always passed as the first parameter for all client side events of WebGrid. (For WebCombo, the Comboid is always passed as the first parameter). Using the Id of the object, the object itself can be obtained

using the utility function `igtbl_getGridById`:

```
<script language='javascript'>
function MyDbClick(gridId, cellId) {
    var oGrid = igtbl_getGridById(gridId);

}
</script>
```

An even easier method for obtaining a reference to the grid object is to use the variable that is automatically generated into the page for it based on the server `ControlId`. In the case of the server control having the name `UltraWebGrid1`, the corresponding client-side variable on the page would be `oUltraWebGrid1`. Again, this variable is generated automatically and can always be referenced as such to obtain the grid object on the client. Our sample code will employ both methods of obtaining `WebGrid` object references.

Once a grid reference is obtained, it can be used to access the entire object model of the grid and its sub-objects. For instance, use the following code to turn on cell updating for the grid and change the style of the main grid HTML element:

```
<script language='javascript'>

function MyDbClick(gridId, cellId) {
    oUltraWebGrid1.AllowUpdate =1; // 0=notset, 1=yes, 2=no, 3=row template only
    oUltraWebGrid1.CellClickAction = 1; // 1=Edit, 2=Row select, 3=Cell select
    oUltraWebGrid1.Element.style.backgroundColor="blue";
    oUltraWebGrid1.Element.style.color="white";
}
</script>
```

The sub-objects of the grid are also available on the main grid object. The `Bands` array contains `Band` objects while the `Rows` array contains the `Row` objects for each row of `Band 0`.

The final task in this introduction to the client-side object model will show you how to iterate through the cells of a column, sum their values and place the value into the first cell of the grid. The JavaScript event handler shown below accomplishes this task.

```
<script language='javascript'>

function MyDbClick(gridId, cellId) {
    var column = igtbl_getColumnById(cellId);
    if(column==null) // If not a column we clicked on, then return
        return;
    var index = column.Index;
    var count = oUltraWebGrid1.Rows.Length; // Get the number of rows in Band 0
    var total = 0;
    for(i=0; i<count; i++) {
        var row = oUltraWebGrid1.Rows.getRow(i); // Get the next row in Band 0
        total += parseInt(row.getCell(index).getValue()); // Get the cell value for this
column, convert it to an integer and add it to the total
    }
    oUltraWebGrid1.Rows.getRow(0).getCell(0).setValue(total); // Place the summed result into
cell(0,0)
}
</script>
```

How Objects Are Created from Rendered HTML and JavaScript

Along with the HTML markup that creates the initial view of UltraWebGrid when a page is loaded, a series of JavaScript arrays and function calls are rendered into the page as well.

As shown in the below code, the arrays contain many property values from the server that are used to implement styles and behaviors on the client. During initialization, the values from the arrays are transferred into objects that can be utilized by client script code as well as by the grid itself.

Each page that utilizes the UltraWebGrid control will have JavaScript generated for it that is similar to the following:

```
<script language="javascript"> <!--
    // AddNewBoxVisible, AddNewBoxView, AllowAddNew, AllowColSizing, AllowDelete, AllowSort,
    ItemClass, AltItemClass, AllowUpdate,
    //CellClickAction, EditCellClass, Expandable, FooterClass, GroupByRowClass, GroupCount,
    HeaderClass, HeaderClickAction, Indentation, NullText,
    //ExpAreaClass, RowLabelClass, SelGroupByRowClass, SelHeadClass, SelCellClass, RowSizing,
    SelectTypeCell, SelectTypeColumn, SelectTypeRow, ShowBandLabels,
    //ViewType, AllowPaging, PageCount, CurrentPageIndex, ImageDirectory, CollapseRowImage,
    ExpandRowImage, CurrentRowImage, NewRowImage, BlankImage,
    //ActiveRect, CultureInfo, RowSelectors, UniqueID
    igtbl_UltraWebGrid1_GridProps = [false,0,2,2,2,2,"UltraWebGrid1-
    ItemClass","",1,3,"",1,"","",0,"UltraWebGrid1-HeaderClass",1,22,"","UltraWebGrid1-
    ExpAreaClass","UltraWebGrid1-RowLabelClass","", "UltraWebGrid1-SelHeadClass","UltraWebGrid1-
    SelCellClass",1,1,2,2,0,0,true,4,1,"/ig_Images/ig_tblMinus.gif","/ig_Images/ig_tblPlus.gif","/
    ig_Images/ig_tblTri.gif","/ig_Images/ig_tblNewrow.gif","/ig_Images/ig_tblBlank.
    gif","UltraWebGrid1_actRect","",|. ",1,"UltraWebGrid1"];
    igtbl_UltraWebGrid1_Bands = [
    // Key, AllowAdd, AllowColSizing, AllowDelete, AllowSort, BandItemClass, BandAltItemClass,
    AllowUpdate, CellClickAction,
    //ColHeadersVisible, ColFootersVisible, CollapseImage, CurrentRowImage, DefRowHeight,
    EditCellClass, Expandable, ExpandImage, FooterClass,
    //GroupByRowClass, GroupCount, HeaderClass, NonSelHeaderClass, HeaderClickAction, Visible,
    IsGrouped, ExpAreaClass, RowLabelClass, SelGroupByRowClass,
    //SelHeadClass, SelCellClass, RowSizing, SelectTypeCell, SelectTypeColumn, SelectTypeRow,
    RowSelectors, NullText
    ["Contacts",0,0,0,0,"","",0,0,1,2,"","", "25px","",0,"","", "0",0,true,
    false,"","UltraWebGrid1-NonSelHeaderClass","", "","", "0,0,0,0,0,0,"]
    ];
    igtbl_UltraWebGrid1_Columns_0 = [
    // Key, DataType, CellMultiline, Hidden, AllowGroupBy, AllowResize, AllowUpdate, Case,
    FieldLen, ButtonsDisplayStyle, HeaderClickAction, IsGroupBy, ItemClass,
    //MaskDisplay, MaskInput, Selected, SortIndicator, Nullable, TabStop, NullText,
    ButtonClass, FooterClass, HeaderClass, SelCellClass, SelHeadClass,
    //Type, ValueList, ValueListClass
    ["LastName","LastName",8,0,false,1,0,0,0,0,0,0,false,"",false,0,"","", "0",
    [], "", ""],
    ["FirstName","FirstName",8,0,false,1,0,0,0,0,0,0,false,"",false,0,"","", "0",
    [], "", ""],
    ["EmailName","EmailName",8,0,false,1,0,0,0,0,0,0,false,"",false,0,"","", "9",
    [], "", ""],
    ["ContactTypeID","Contact Type",3,0,false,1,0,0,0,0,0,0,false,"",
    false,0,"","", "5", "", [{"1","Buyer"}, {"2","Seller"}, {"3","Manager"},
    {"4","Assistant"}], "", ""],
    ["Single","Single",8,0,false,1,0,1,0,0,0,0,false,"",false,0,"","", "3", "", [{"1","Single"}],
    ["Save<img src='go.gif' align=absmiddle>","Save<img src='go.gif' align=absmiddle>",8,0,
    false,1,0,0,0,0,1,0,false,"",false,0,"","", "7", "", [{"1","Save"}]]
    ];
    igtbl_UltraWebGrid1_Events = [
```

```

    // AfterCellUpdate, AfterColumnSizeChange, AfterEnterEditMode, AfterExitEditMode,
    AfterRowActivate, AfterRowCollapsed, AfterRowDeleted, AfterRowExpanded,
    //AfterRowInsert, AfterRowSizeChange, AfterSelectChange, BeforeCellChange,
    BeforeCellUpdate, BeforeColumnSizeChange, BeforeEnterEditMode, BeforeExitEditMode,
    //BeforeRowActivate, BeforeRowCollapsed, BeforeRowDeleted, BeforeRowExpanded,
    BeforeRowInsert, BeforeRowSizeChange, BeforeSelectChange, ClickCellButton, CellChange,
    //DbClick, EditKeyDown, EditKeyUp, InitializeLayout, InitializeRow, KeyDown, KeyUp,
    MouseDown, MouseOver, MouseOut, MouseUp, ValueListSelChange
    [ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],
    [ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "HideMe",0],
    [ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ "",0],[ ""],
    ];
--> </script>

```

The first array is created for the grid as a whole. It contains values for many properties of the DisplayLayout object on the server. There is also an array of bands created and an array of columns for each band. The last array contains entries for clientside events.

Following the HTML markup for the grid itself is some additional JavaScript which serves to initialize the grid and create the primary objects for client-side manipulation.

```

<script language="javascript"> <!--
    var igtbl_UltraWebCombolxctl0 = igtbl_initGrid("UltraWebCombolxctl0");
    igtbl_UltraWebCombolxctl0.gridElement.parentElement.scrollTop=0;
    igtbl_updatePostField("UltraWebCombolxctl0");
    igtbl_gridState['UltraWebCombolxctl0'].GridIsLoaded=true;
--> </script>

```

The `igtbl_initGrid()` function creates a Grid object along with Band and Column objects by transferring the values from the JavaScript arrays that were rendered above into JavaScript object properties. It also creates an object that contains the event information and attaches it to the grid object.

If the grid is named 'UltraWebGrid1' on the server then the above code enables client-side code scenarios like the following:

```

var gridObject = igtbl_getGridById('UltraWebGrid1');
var Band = gridObject.Bands(0);
Band.AllowAdd = true;
var Column = Band.Columns(2);
Column.AllowResize = 0;

```

How Objects Are Created Dynamically on the Client

The topic [How Objects Are Created from Rendered HTML and JavaScript](#) describes how, as the page is loaded into the browser, objects are created on the client for the UltraWebGrid control, Bands and Columns. While these objects are created for the control and each Band and Column within it, there are initially no objects created for Rows or Cells within UltraWebGrid. However, objects for these entities can be created dynamically as they are needed throughout the life cycle of the page within the browser.

Several utility functions are available that dynamically create objects for rows and cells. `igtbl_getRowById(cellId)` and `igtbl_getCellById(cellId)` obtain Row and Cell objects respectively.

An array of Cell objects is returned by the **Row.getCells()** method. An array of Row objects is returned by the **Row.getRows()** method.

Using The Object Model To Navigate Grid Objects

Client-side script can navigate between bands, rows, columns and cells in the grid using utility functions, objects and object arrays as illustrated in the following code examples. This code is based on a client-side event handler for the DbClick event.

The WebGrid help file contains a complete reference for all properties and methods that are available on each of the client-side objects.

The following JavaScript code sample details how to get a reference to the grid, rows and cells. The values of cells can be set from the client-side as well as other properties on other objects. For example, the last section of code shows how to set the AllowResizing property of a column.

```
// Obtain a reference to the grid objects:

function DbClick(gridId, cellId) {
// Obtain a reference to the grid object
var oGrid = igtbl_getGridById(gridId);

// Obtain a reference to the cell that was double clicked
var oCell = igtbl_getCellById(cellId);

// Set the contents of the cell to a new value
oCell.setValue("NewValue");

// Obtain a reference to the row in which the cell was double clicked
var oRow = igtbl_getRowById(cellId);

// Navigate to the previous row and set the contents of the third cell to a new value
oRow = oRow.PrevSibling();
if(oRow != null) {
    oCell = oRow.getCell(2);
    oCell.setValue("New Cell Value");
}

// Obtains the current cell object again and references the associated column to turn off
resizing
oCell = igtbl_getCellById(cellId);
oCell.Column.AllowColResizing = 1;

return true;
}
```

The *oGrid.Bands* array contains all Bands on the client. The *oBand.Columns* array contains all column objects within the band. The *oCell.Column* and *oCell.Band* properties contain references to the Column and Band that the cell object belongs to.

Note Many properties of the Band and Column objects are informational only. While all properties are writeable through JavaScript on the client, changing the values will have no immediate effect on the client. For example: Changing the *Column.ItemClass* property will not result in all cells of the column immediately changing to the new class. Only cells that need to have their class set back to the *ItemClass* value will get updated as when they are no longer the current or selected cell. This applies to all CSS class properties of objects. Furthermore, no property changes effected on the client will be updated on the server grid object.

What's New in Version 3

New for 3.0.20041.11

The 3.0.20041.11 update to Infragistics.WebUI.UltraWebGrid.v3.dll (released in February 2004) adds these key enhancements to the WebGrid's overall functionality:

Export to Excel

You can now easily export the contents of a WebGrid to an Excel spreadsheet. Using the UltraWebGridExcelExporter element, you can send the entire contents of a WebGrid to a downloadable spreadsheet. Follow the [Export to Excel Tutorial](#) to learn how to implement this feature.

Client-Side Activation Object Changed

The WebGrid no longer uses floating DIV's for client activation. The Grid Object has a new **Activation** property object. **Grid.ActiveCell** and **Grid.ActiveRow** are obsolete, you should use then **grid.oActiveRow** and **grid.oActiveCell** objects instead.

Section 508 Compliance

The **WebGrid**, **WebMenu** and **WebTree** controls now provide the ability to render Section 508 compliant html to the browser. The **Section508Compliant** property under the **DisplayLayout** node of the Section 508 compliant elements is set to **False** be default. Setting this property to **True** enables the Section 508 compliant functionality.

⊕ RowStyleDefault	Has Data
ScrollBar	Auto
ScrollBarView	Both
Section508Compliant	False
⊕ SelectedGroupByRowStyleDefault	
⊕ SelectedHeaderStyleDefault	
⊕ SelectedRowStyleDefault	
SelectTypeCellDefault	None
SelectTypeColDefault	None
SelectTypeRowDefault	None
StationaryMargins	No
⊕ Strings	
TabDirection	LeftToRight

[Edit Layout](#), [Edit Columns](#), [Edit Rows](#), [Property Builder](#), [Load Layout](#), [Save Layout](#), [Enable HTML Intellisense](#)

Section 508 Compliance involves making sure that your application can be used effectively by end-users with disabilities.

This primarily means that you must ensure some level of accessibility to:

- Users with low vision or no vision who may be using assistive technologies such as screen magnifier or screen reader software.
- Users with limited movement who may be using alternative input methods instead of the standard keyboard and mouse combination. This might include simply a keyboard with no mouse, or some kind of alternate pointing device that does not have the precision of a mouse.

(Section 508 compliance also includes considerations for users with hearing disabilities. However, audio considerations are outside the scope of this documentation, because Infragistics products primarily provide visual interfaces.)

If you are developing applications for a United States Federal agency, accessibility is a requirement of your application under Section 508 of the Rehabilitation Act of 1973. For more information, you can refer to the governmental web site <http://www.section508.gov/>. Keep in mind that Infragistics only provides the tools for you to use. It is ultimately the responsibility of the developer to design an interface that is usable by people with disabilities.

Column Sort Event

The Shift key state is available from within the **SortColumn** event of **WebGrid** as an **EventArgs** parameter names **e.Shift**.

```
Private Sub UltraWebGrid1_SortColumn(ByVal sender As Object, _  
    ByVal e As Infragistics.WebUI.UltraWebGrid.SortColumnEventArgs) _  
    Handles UltraWebGrid1.SortColumn
```

e.ShiftKey

End



Public Property ShiftKey() As Boolean
Shows if the shift key is pressed while the header is clicked.

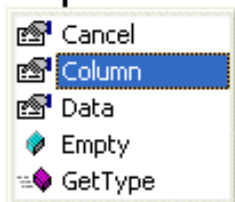
Column Move Event

A server-side **ColumnMove** event has been added. This event gives you more granular control over the action you can take in code-behind when a column is moved in the WebGrid.

```
Private Sub UltraWebGrid1_ColumnMove(ByVal sender As Object, _  
    ByVal e As Infragistics.WebUI.UltraWebGrid.ColumnEventArgs) _  
    Handles UltraWebGrid1.ColumnMove
```

e.

End sub



Public ReadOnly Property Column() As Infragistics.WebUI.UltraWebGrid.UltraGridColumn
Returns the Column object associated with the event.

Embeddable Editor Support for WebDataInput Editors

The new **WebDataInput** editors include **WebTextEdit**, **WebMaskEdit**, **WebNumericEdit**, **WebCurrencyEdit** and **WebDateTimeEdit**. These editors can be embedded into cells of the WebGrid, providing a very robust way to manage how users input data directly in the WebGrid. See the Embedding [Web Text Editors in WebGrid](#) tutorial for information on implementing this functionality.

The **WebDataInput** elements can also be used as stand-alone editors on an ASPX page. Each of the editors supports an extensible client-side object model, full masking support, as well as server-side properties, methods and events. The WebDataInput help file, with samples, can be found in the default installation directory in the **WebDataInput** folder.

New Pager Features

The paging options for WebGrid have been enhanced to include the the following PagerStyleMode options:

PagerStyleMode	Description
Numeric	Pages are represented by numbers
PrevNext	Pages are represented in the Prev Next view.
CustomLabels	User defines how pages are represented.
QuickPages	Allow to set quantity of quick pages that are accessible from the current page.
ComboBox	Pages are represented in the combo box style.

StationaryMargins

StationaryMargins now work in **ReadOnly** mode.

WebCombo XP Look and Feel

Two properties have been added to WebCombo:

- DropImageXP1
- DropImageXP2

Two new images corresponding to these new properties, **ig_cmboDownXP1.bmp** and **ig_cmboDownXP2.bmp**. The combo looks into the user agent string and if the version of Windows is 5.1 or higher this set of images is used. The border style is defaulted to **Solid**. The border color is defaulted to **LightGray**.

New for Version 2.1

Version 2.1 of the UltraWebGrid (released October, 2003) adds the following key features to the product:

Improved Design-Time Interface & New Styles

There are an even greater number of pre-defined styles for you to choose from. The control has also gained the following design-time enhancements:

Design-Time HTML Intellisense

To enable design-time HTML Intellisense, bring up the control's context menu. You will see the option "Enable HTML Intellisense". Choosing this option puts an extra attribute in the WebForm's body tag. This attribute will allow Intellisense to work at design-time, in the HTML view of the webform. When you are done working with the HTML, you can turn off this option and remove the attribute by again bringing up the context menu and unchecking the "Enable HTML Intellisense" option.

Client-Side Events Editor

Adding a handler for a client-side event is now just as easy as adding a server-side event handler. Clicking on the **ClientSideEvent** property in the Visual Studio property sheet at design-time will drop down a list of JavaScript functions currently defined on your WebForm. There will also be an option to add a new event handler. When this option is chosen, a new JavaScript function stub will automatically be generated for you and added to your WebForm. You will automatically be brought into the new function, and you can immediately start writing code to handle the event.

Integrated Date Dropdown and Enhanced Date Formatting

You can now use the DateChooser control (as found in the new WebSchedule control) as an integrated date-selection dropdown for grid columns that permit the entry of date data. The dropdown provides data masking for entered data and displays a monthly calendar when dropped down for point-and-click date selection.

Better Paging

Support for paging has been enhanced to offer better functionality when displaying data in a paged (as opposed to scrolling) view.

Better Templating

Control templates now enjoy better support, giving you even more control over customizing the grid's interface.

Enhanced Netscape Compatibility

Compatibility with the latest Netscape/Mozilla series of browsers has been improved, making the grid more suitable for cross-browser environments.

What's New in Version 2

New for 2.1.20033

The 2.1.20033 update to UltraWebGrid (released in October 2003) adds these key enhancements to the WebGrid's overall functionality:

Design-Time HTML Intellisense

To enable design-time HTML Intellisense, bring up the control's context menu. You will see the option "Enable HTML Intellisense". Choosing this option puts an extra attribute in the WebForm's body tag. This attribute will allow Intellisense to work at design-time, in the HTML view of the webform. When you are done working with the HTML, you can turn off this option and remove the attribute by again bringing up the context menu and unchecking the "Enable HTML Intellisense" option.

Client-Side Events Editor

Adding a handler for a client-side event is now just as easy as adding a server-side event handler. Clicking on the **ClientSideEvent** property in the Visual Studio property sheet at design-time will drop down a list of JavaScript functions currently defined on your WebForm. There will also be an option to add a new event handler. When this option is chosen, a new JavaScript function stub will automatically be generated for you and added to your WebForm. You will automatically be brought into the new function, and you can immediately start writing code to handle the event.

Version 2 of the UltraWebGrid contains a series of enhancements that further expand the ability to use rich-client interface features in web-based (thin-client) applications. Among the improvements to Version 2 are the following:

Column/Row Templates

WebGrid supports ASP.NET template technology for both columns and rows of the grid. With column templates, the cells of a column can be custom rendered using ASP.NET controls to represent data values. With row templates, developers can create custom forms that drop down from a selected row. This allows complex data entry scenarios to be implemented easily.

[See more help on Column and Row Templates](#)

WebCombo: Multi-Column Drop-Down Control

The WebCombo control is an editable, multi-column combo box for web developers. Based on the WebGrid, WebCombo allows users to enter values by selecting from a list of items or by typing directly into WebCombo's edit box.

WebCombo Features

- Bound and unbound modes
- Multiple column display
- Support for hidden columns
- Client-side events, object model and API
- Editable and non-editable style
- Column templates
- Column and row resizing
- Column sorting

- Full CSS support for column headers, row labels and cells
- Design-time support for adding rows and columns, selecting styles and predefined layouts, and viewing combo dropdown.

[See more help on the WebCombo control](#)

Column Moving and Sorting

Columns headers can be dragged and repositioned on the client so that the new column ordering can be reflected when the control is re-rendered from the server.

Column headers can also be used to sort individual columns on the client without requesting data from the server.

[See more help on Client-Side Column Moving and Sorting](#)

Cell Merging

Cells with common values can be displayed so that the value appears only once, and the area of the cells is merged into a single area. Any adjacent cells can also be merged on a manual basis using cell properties.

[See more help on Cell Merging](#)

Stationary Column Headers

Column headers and/or footers can now remain stationary as rows are scrolled. This assists the user with keeping the displayed data in context.

[See more help on Stationary Headers and Footers](#)

Load and Save Layouts to XML

The complete look and behaviors of WebGrid can be saved to disk as an XML file and reused again and again, both at design-time and run-time.

[See more help on Loading and Saving Layouts](#)

Read-Only Mode

Read-only mode is useful for displaying data on the client when there is no need for the user to update that information. By removing the client-side functionality required to facilitate data updating, read-only mode conserves bandwidth and speeds page loads.

Two different levels of read-only functionality are supported. Level One is a pure read-only mode that renders the data with absolutely no JavaScript on the client. Level Two uses limited JavaScript to enable minimal behavior on the client, such as row expansion and column/row resizing.

[See more help on Read-Only Mode](#)

Load on Demand

To reduce network traffic and to increase viewstate efficiency, bands can be populated on an as-needed basis. Two styles of new load-on-demand functionality are supported: Manual and Automatic. With manual load-on-demand, the application must respond to the **DemandLoad** event by populating the rows of the expanded band. Automatic load-on-demand is used in conjunction with data binding. With the Automatic style, UltraWebGrid populates only those nodes that are expanded at the time that the databinding takes place. Other nodes are populated only when the user expands them.

Using either of these techniques eliminates the need for data in non-expanded rows to travel down to the browser. The benefit of load-on-demand is that the developer can control how much data is sent to the client and how often the server is called for data.

Enhanced Formatting and Validation

Data entry on the client has been enhanced to allow for data formatting and validation as cell values are entered.

Enhanced Client-Side API

The client-side object model of the UltraWebGrid has been significantly enhanced with greater support for objects, methods, properties and events. Additional capabilities have been added for accessing and manipulating the WebGrid on the client without the need for server post-backs.

Expanded Documentation

The help system for this version of UltraWebGrid has been greatly expanded. New In-Depth Tutorial topics provide guided learning with illustrated examples. New Object Trees visually display the hierarchy of the client and server side object models. Multiple Task-Based Help topics have been added. The API reference now contains code snippets to illustrate key points, and has been generally expanded with more comments and descriptions. And a new Documentation Atlas topic provides multiple ways to access key areas of the documentation. and the Table Of Contents has be re-organized.

[See a list of all the additions to the V2 Help System](#)

Introduction - WebCombo DropDown Control

The WebCombo control is a multi-column editable dropdown component. The DropDown portion of the control is a WebGrid that is displayed beneath the data value portion at the top of the control. The **Editable** property of WebCombo is used to determine whether the top portion of the control allows values to be typed in with the keyboard.

Note It is very important to remember to manually add the following Register directive to the top of the *aspx* file prior to configuring columns or rows of the WebCombo element.

```
<%@ Register TagPrefix="igtbl" Namespace="Infragistics.WebUI.UltraWebGrid"
    Assembly="Infragistics.WebUI.UltraWebGrid.v2" %>
```

Add this line just above the tag prefix registration for the WebCombo control that the Visual Studio environment adds when the control is placed on the form. If this Register directive is not added, the Visual Studio environment will not be able to resolve tags on the page that reference objects in the UltrawebGrid assembly. These objects include Bands, Columns, Rows and Cells.

An easier way to accomplish this same task is to drop the UltraWebGrid on the page and then immediately delete it. This will cause the Register TagPrefix directive to be added for you automatically.

The Webcombo control can be bound to any collection of objects that supports the basic data binding interfaces of ADO.NET. It can also function in unbound mode in much the same manner as UltraWebGrid. Right-clicking on the control at design-time displays a menu of options for editing the Layout of the DropDown object as well as configuring the columns and/or rows of the dropdown. The "Show DropDown" menu item allows the dropdown to be visible at design time.

The **DataSource** and **DataMember** properties can be set at design time to a DataSet object on the page. Adapter.Fill() and DataBind() must still be called from the appropriate point in the code behind file in order for the data binding to take place at run-time.

One of two properties can be used to determine which column of the DataSource will be used to display items in the value area of the WebCombo. The **DataTextColumn** property selects a column based on the key value of the column. The **DataValue** property of WebCombo is used to obtain the value that is selected or entered into the value area at the top of the WebCombo control.

The **SelectedIndex** and **SelectedRow** properties of WebCombo determine which item is currently selected and displayed in the value area at the top of the control. The **SelectedIndex** property selects the row by its index and the **SelectedRow** property selects the row that is passed to the combo.

WebCombo Events

The following events are fired by WebCombo on the server:

InitializeLayout - This event is fired when the control has been bound to the datasource and columns have been configured, but rows of data have not been loaded yet. The *LayoutEventArgs* parameter contains a reference to the UltraWebGrid.Layout object that is being initialized.

InitializeRow - This event is fired for each row that is bound to the DropDown of the WebCombo. The *RowEventArgs* parameter contains a reference to the UltraGridRow object that is being initialized.

SelectedRowChanged - This event is fired when the user changes the selected row in the dropdown and updates the value portion of WebCombo. The *SelectedRowChangedEventArgs* parameter contains information about the row that has been changed.

In addition to these three events that are fired on the server, there is an assortment of client-side events that can be handled from JavaScript on the client. The ClientSideEvents object of WebCombo contains the properties that can be set for these events. The string value that is set on the property identifies a JavaScript function that will be called on the client when the event fires. All events pass the element id of the WebCombo control as the first parameter.

Using ReadOnly Mode

The `DisplayLayout` object contains a property called **ReadOnly** which controls the amount of JavaScript and HTML rendered to the client machine. WebGrid supports two levels of `ReadOnly` mode: `LevelZero` contains absolutely no JavaScript functionality and is designed to render the absolute minimum amount of HTML to achieve the appearance and style characteristics specified. With `LevelZero` `ReadOnly` mode, only grid scrolling is supported on the client. `LevelOne` contains a minimum amount of JavaScript for some basic client-side functionality. Scrolling, row expansion and collapse and column resizing are supported.

When the **ReadOnly** property is set to either of its two level values, there is no client-side object model or client-side event support.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ReadOnly = ReadOnly.LevelZero
```

In C#:

```
UltraWebGrid1.DisplayLayout.ReadOnly = ReadOnly.LevelZero;
```

Loading and Saving Layouts

The style and behavior properties of the `DisplayLayout` object can be loaded from an XML source at run-time as well as at design-time. This capability allows the major stylistic and behavioral aspects of the grid to be saved as XML and reused at a later time.

Loading and saving layouts at design-time

At design-time layouts can be loaded from a predefined selection of XML files that are stored in the `Layouts` directory below the WebGrid installation directory.

1. To load a layout, right-click the WebGrid and select the Load Layout menu item. All XML files in the layout directory are displayed and can be selected. For each XML file that has a corresponding bitmap file with the same root name and a `.bmp` extension, the bitmap will be displayed to the right of the layout selection list.
2. Choose a file and click "Apply" to see the results of the XML applied to the grid.

Similarly, layouts can be saved to xml files at design-time as well.

1. To save a layout, right-click the grid and select the Save Layout option. A directory listing of the Layouts directory will be displayed.
2. Enter a filename that will be used to save the layout settings. You can also choose an existing file - the current layout settings will be saved, overwriting the settings in the chosen file.

Loading and saving layouts at run-time

Loading layout XML at run-time can be accomplished with the `UltraWebGrid.DisplayLayout.LoadLayout` method. This method has two overloads: One that uses an `XmlTextReader` object for the Layout source and one that uses an xml string as the Layout source.

In Visual Basic:

```
Dim xmlReader As New System.Xml.XmlTextReader("MyLayoutFile.xml")
UltraWebGrid1.DisplayLayout.LoadLayout(xmlReader, True, True, True, True)
xmlReader.Close
```

In C#:

```
System.Xml.XmlTextReader xmlReader = new System.Xml.XmlTextReader("MyLayoutFile.xml");
UltraWebGrid1.DisplayLayout.LoadLayout(xmlReader, true, true, true, true);
xmlReader.Close();
```

The second and third parameters indicate whether style properties and behavior properties should be loaded respectively.

Saving layout XML at runtime can be accomplished with the `UltraWebGrid.DisplayLayout.SaveLayout` method. This method has three overloads: One that uses an `XmlTextWriter` object for the Layout source, one that takes the name of a file to use for saving the XML to disk and one that returns an XML string containing the XML data.

In Visual Basic:

```
Dim xmlWriter As New System.Xml.XmlTextWriter("MyLayoutFile.xml", Nothing)
UltraWebGrid1.DisplayLayout.SaveLayout(xmlWriter, True, True, True, True)
xmlWriter.Close
```

```
' Save Layout as a string
Dim xmlLayout as String
```



```
xmlLayout = UltraWebGrid1.DisplayLayout.SaveLayout(True, True, True, True)
```

In C#:

```
System.Xml.XmlTextWriter xmlWriter = new System.Xml.XmlTextWriter("MyLayoutFile.xml", null);  
UltraWebGrid1.DisplayLayout.SaveLayout(xmlWriter, true, true, true, true);  
xmlWriter.Close();
```

```
// Save Layout as a string  
string xmlLayout = UltraWebGrid1.DisplayLayout.SaveLayout(true, true, true, true);
```

The last two parameters of the **SaveLayout** method indicate whether or not to save style and behavioral properties respectively.

Row and Column Templates

Templates provide a means for using child controls to perform various tasks in place of the UltraWebGrid's native editing functionality. You can use templates to extend the power of the UltraWebGrid, making it the centerpiece of a highly-customized data-entry application. With templates, the users of your program work with a familiar grid interface for general data viewing and organization, plus they gain the advantage of working with a custom-designed mini-application for specialized tasks.

Column Templates

In UltraWebGrid, columns can contain templates within the header area, within footers, and within each cell of the column. To add a template to a column:

1. Right-click on the control and choose "Edit Columns" from the context menu.
2. Select the Column to be templated and check the Templated box beneath the column list. This creates a TemplatedColumn object that replaces the pre-existing column.
3. Exit the Column Editing form and right-click the control to select the Edit Templates item. Any columns that have been marked as Templated Columns will appear in the Edit Templates submenu. Select a column to edit the templates for that column.

Each column can have a Header, Footer and Cell template associated with it. The Cell template will be repeated for each cell within the column. By editing the XML within the .aspx file directly, the controls within the cell template can be bound to the data of the cell.

In the below sample the value property of the asp textbox control is assigned the Text property of the current cell container for the template. This code gets executed for each row being databound to the grid.

```
<CellTemplate>
<asp:TextBox id="TextBox1" value="<%= Container.Text %>" runat="server"></asp:TextBox>
</CellTemplate>
```

In addition, two other properties are available from the containing cell that are useful in various situations. They are Container.Value and Container.FormattedText.

The Header Template has several container properties available as well. They are Container.HeaderText and Container.SortIndicator. **SortIndicator** is a URL string. The Footer Template has Container.FooterText and Container.FooterSummary properties available. These properties are all of type string.

Row Templates

Row editing templates provide the ability to create a custom form for displaying and editing the cells of a row. The controls inside the template can be populated with the data of the row using HTML attributes that identify which column of the row relates to each template control.

Each Band of the grid can have a template for editing the rows of the band. At run-time, the row editing template is displayed when the row selector is clicked, or it can be invoked from JavaScript code.

Before row templates can be displayed and used within WebGrid, the **AllowUpdate** property of the Band or **DisplayLayout** must be set to one of two values. If it is set to Yes, then cells will be enterable directly by clicking on them and typing into them. They can also be updated by displaying the RowTemplate in response to a RowSelector click or javascript invocation. If the **AllowUpdate** property is set to RowTemplateOnly, then editing of cells directly is not permitted, but cell values can be updated through the row template.

Creating Templates

To add a row editing template to a Band, right-click the control and select the Edit Templates menu item. The submenu displayed will show one Row Editing template selection for each band of the grid that is configured. By default, a row editing template for Band 0 is always available. The first time the template is edited, you will be asked if you would like to populate the template with a label and textbox for each column of the band. There will also be an OK and Cancel button placed at the bottom of the template form.

The Band object contains a **RowTemplateStyle** property that allows you to configure the appearance and style of the row editing template as a whole.

Once the initial template has been created for you, you can drag and drop the controls within the template as well as add new controls and delete existing ones from the template. Visual Studio does not allow you to edit the template in Grid Layout mode. You must use Flow Layout to position the controls. You can use a table or Panel controls to gain further control over positioning. You can also edit the controls in the template by editing the XML of the .aspx file directly. This is required in the case of attaching server-side event handlers to ASP.NET controls in the template.

Transferring Data

To associate a control in the template with a column in the Band, a `ColumnKey` or `ColumnNo` attribute is added to the template control. At runtime, the controls in the template are scanned for these attributes when the template is about to be displayed. For each control with one of these attributes, a client-side **TemplateUpdateControls** event is fired. This provides the opportunity to perform a conversion of the data in a grid cell to the format of the template control. If the template control is a text box, the default handling will make the conversion automatically.

When the template editing form is about to be closed, the reverse process occurs. The controls of the template are once again scanned for the `ColumnKey/ColumnNo` attribute in order to convert the data in the template controls back to values for the row cells. In this case, the **TemplateUpdateCells** event is fired for each control in order to perform the update. Once again, for Text Box controls, the default processing will take the text from the control and put it into the corresponding grid cell automatically.

When the controls in the template are added automatically by the WebGrid Designer, they are given appropriate `ColumnKey/ColumnNo` attributes in order to make the link automatic.

Client Side Templating Events

In addition to the two events described above that are involved with the transfer of data between cells and controls, four other client side events can be used in response to template related events.

BeforeRowTemplateOpen is fired prior to the row template display. Returning a non-zero value for this event will prevent the template from opening. **AfterRowTemplateOpen** is fired after the template has been loaded and displayed. This event allows and final positioning and initialization to occur. **BeforeRowTemplateClose** is fired prior to the closing of the row template. Returning a non-zero value for this event will prevent the template from closing. **AfterRowTemplateClose** is fired after the template has been unloaded and closed. This event allows final edits and post editing behavior to occur.

Using Hidden Columns On The Client

Columns that are hidden on the server will still be rendered to the client but will not be visible. Columns that are not used from the DataSource can be deleted from the Columns collection at design-time or at run-time. Hidden columns on the client can be accessed from JavaScript using the client-side object model. All columns on the client can be hidden and shown using the setHidden() method of the Column object.

Client-Side Column Moving and Sorting

Enable Client-Side Column Moving

Columns on the client can be dragged and dropped by the user to re-order them. Once a column is dropped at a new location the page is posted back to the server where the data is rendered in the new column order.

To enable Column Moving on the client, set the `DisplayLayout.AllowColumnMovingDefault` property to `OnServer` or `OnClient`. In both cases the user can drag and drop a column on the client.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.AllowColumnMovingDefault = AllowColumnMoving.OnServer
```

In C#:

```
UltraWebGrid1.DisplayLayout.AllowColumnMovingDefault = AllowColumnMoving.OnServer;
```

Client-Side Sorting

Data can be sorted on the client-side by the end user without requiring a postback to the server. This is done through the column header interface, which displays an image that indicates whether the column is being used to sort the data, and whether the data in that column is sorted in ascending or descending order. Users click in the column headers to dynamically change the sorting of the grid data.

When a column is sorted on the client-side, the control adds it to an array of columns that are being used as sort criteria. Through code, you can manage this process by manually adding columns to the sorted column array, by initiating the sorting process using the columns currently in the array, or by sorting on a specified column. The `sort`, `sortColumn` and `addSortColumn` methods and the `BeforeSortColumn` and `AfterSortColumn` client-side events of the [grid object](#) are used to programmatically access the client-side sorting features of the control.

You can choose whether to allow multi-column sorting, or limit the user to using one column at a time as the sort criteria.

Column sorting can be performed on the client by setting the `DisplayLayout.AllowSortingDefault` to `OnClient`:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.AllowSortingDefault = AllowSorting.OnClient
```

In C#:

```
UltraWebGrid1.DisplayLayout.AllowSortingDefault = AllowSorting.OnClient;
```

Hide Cell Text Overflow and Display Ellipsis

The wrapping of text within cells and the display of ellipses for overflowed cells is controlled using properties of the Style object. To prevent text wrapping set the Style.**Wrap** property to False. To show ellipses for cells that overflow, set the Style.**TextOverflow** property to Ellipsis.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.RowStyleDefault.Wrap = False  
UltraWebGrid1.DisplayLayout.RowStyleDefault.TextOverflow = Infragistics.WebUI.UltraWebGrid.  
TextOverflow.Ellipsis
```

In C#:

```
UltraWebGrid1.DisplayLayout.RowStyleDefault.Wrap = false;  
UltraWebGrid1.DisplayLayout.RowStyleDefault.TextOverflow = Infragistics.WebUI.UltraWebGrid.  
TextOverflow.Ellipsis;
```

Cell Merging

Cell Merging is the ability to have the size of a single cell encompass the area of several cells. Cell merging can be vertical, horizontal, or both. In all cases, the cell located at the top left position is the cell whose contents and appearance are used to render the area of the merged cell.

UltraWebGrid supports two different types of cell merging: automatic and manual. With automatic cell merging, adjacent cells in a column that have common values are merged to form a single cell that displays the value in common. Automatic cell merging always results in a vertical merged cell area. Automatic cell merging is turned on at the column level using the **MergeCells** property of the UltraGridColumn object. This property can be set at design-time and run-time.

With manual cell merging, any cell can be designated to occupy any number of rows and columns to form a larger area in which the contents of the merge cell will be displayed. Manual cell merging is turned on by using the **RowSpan** and **ColSpan** properties of the UltraGridCell object.

Expanded Help File

The help system in V2 of the UltraWebGrid has been greatly expanded over the V1 help. In addition to the new material about the new functionality of the control, a lot has been added that covers both old and new functionality.

The [What's New In V2 topic](#) will direct you to the areas of the help file that have been updated to cover the new control features. The current topic will highlight some of the overall improvements to the help system which are not related to new control functionality.

This topic also contains the following sections:

- [Table Of Contents](#)
- [Tutorials](#)
- [Task-Based Help](#)
- [Code Snippets](#)
- [Other Additions](#)

Table Of Contents

The Table Of Contents for the help file has been completely reorganized for this release. Help system topics are now grouped into a small number of high-level categories according to the type of information presented, providing fast access to the type of information that you want. The TOC is also larger, with the addition of new Object Trees, which are structures in the TOC that illustrate the hierarchical relationships present in the object models of the controls themselves.

Tutorials

Thirty-six new Tutorial topics have been added under the "In-Depth" category of the Table Of Contents heading "Tutorials: Learning To Use WebGrid/Combo". The tutorials are illustrated and follow a common format to speed learning.

Task-Based Help

Seventeen new Task-Based Help topics have been added to this release, and Task-Based Help is now grouped together under a common heading and broken down according to the type of task you are trying to perform.

The following is a list of the Task-Based Help topics added in this release:

- | | |
|---|---|
| <ul style="list-style-type: none">• Cell Merging• Client-Side Column Moving and Sorting• Handling Edit Mode on the Client-Side• Client-Side Object Looping• Client-Side Searching (Find/Find Next)• Suspend Data Updates on the Client-Side• Confirm Data Deletion With Message Box• Hide Cell Text Overflow and Display Ellipsis• Loading and Saving Layouts | <ul style="list-style-type: none">• Making Column Headers and Footers Stationary• Manually Drop Down the WebCombo• Popup a WebMenu on Right-Click Over Cell• Row and Column Templates• Using Hidden Columns On The Client• Using ReadOnly Mode• WebCombo With ValueList• WebCombo As DropDown List |
|---|---|

Code Snippets

Key topics in the API reference section of the help system have had code snippets added to further illustrate how to use the item being documented. Most of these snippets are concentrated in the Events section of the API reference. Check out the Example sections of the [AddRow](#), [AddRowBatch](#), [InitializeRow](#) and [InitializeLayout](#) topics to see an example of the added documentation.

Other Additions

A new [Documentation Atlas topic](#) provides multiple overviews (or "maps") of the help system content. This topic assists you in using the help system, and makes it easier to find both commonly-used topics and material that might be overlooked.

A number of organizational topics have been added, such as the topics that appear when you click the top-level node of an Object Tree. (Hint: These topics reproduce the Object Tree in a printable format.)

Introduction - Release Notes

The following notes apply to this release of UltraWebGrid. This information can also be found in the README file for the product.

Upgrading to Version 3

Version 3 of UltraWebGrid appends a **.v3** to the end of the assembly name to help insure proper versioning in Visual Studio and to enable clean side-by-side execution with other versions of the product. To upgrade an existing project to version 3, follow these steps:

Note Before upgrading your project you should *make a backup copy* of all files in the project.

1. After opening your project, click on Project References and remove the entry for Infragistics.WebUI.UltraWebGrid.dll. Right-click to add a reference and select Infragistics.WebUI.UltraWebGrid.v3.dll from the list of available .NET components.
2. If you attempt to open a web application (.aspx) form containing UltraWebGrid, you will see the controls displayed as an error. To fix this, click the HTML tab at the bottom of the form and edit the .aspx file as text. In the `<%@ Register TagPrefix` directive at the top, change the Assembly attribute from `Infragistics.WebUI.UltraWebGrid` to `Infragistics.WebUI.UltraWebGrid.v3`.

Upon completing this change you should **immediately save the .aspx file and close it.**

Do not switch back to design mode and make changes to the file!

Once the file has been saved and closed it can be reopened and the WebGrid v3 will be loaded in place of the previous version.

3. Open the file called `licenses.licx` in your project directory with notepad and change the reference from `Infragistics.WebUI.UltraWebGrid.dll` to `Infragistics.WebUI.UltraWebGrid.v3.dll`. Save the `licenses.licx` file.
4. Delete the file `Infragistics.WebUI.UltraWebGrid.dll` from the bin directory of the project so that only the `Infragistics.WebUI.UltraWebGrid.v3.dll` file remains. This will eliminate runtime errors due to multiple references to objects.

UltraWebGrid Client-Side Object Tree

The WebGrid Client-Side Object Tree provides a drill-down interface based on the object model of the WebGrid control as it exists on the client within the browser. Beginning with the Grid itself at the highest level, the various sub-objects can be traversed and navigated so that a clear picture of the object relationships can be gained.

[WebGrid Client-Side Object Tree](#)

- └─ [Introduction and Overview](#)
- └─ [Utility Functions](#)
- └─ [Grid Object](#)
 - └─ [Bands Property](#)
 - └─ [Band Object](#)
 - └─ [Row Object](#)
- └─ [Band Object](#)
 - └─ [Columns Property](#)
 - └─ [Column Object](#)
- └─ [Column Object](#)
 - └─ [ValueList Property \(array\)](#)
- └─ [Row Object](#)
 - └─ [Rows Collection](#) (Child Rows)
- └─ [Rows Collection](#)
 - └─ [Row Object](#)
- └─ [Cell Object](#)
- └─ [ExpandEffects Object](#)

Client-Side Object Model - Utility Functions

Introduction

The utility functions described here are used by the JavaScript code of the UltraWebGrid Control. The purpose of exposing and documenting them is to allow developers to put as much logic as possible on the client machine in order to obtain gains in both performance and functionality.

Many of these Utility functions are involved with more than simply returning a property value. They are concerned with 'resolving' the property value according to an ordered set of precedence rules. The rules of precedence are straightforward: Check the local value first, if that value is 0, meaning notset, check the next most local value. This may be the Row, Column, Band or Grid. Repeat the procedure until either a value is found, or the Grid object default is encountered and applied.

HTML ID Formats

Many functions and properties are based on the Id of elements within the page. The Ids of elements is based upon a method of identifying Table and Cell objects uniquely as to owner and position within the Document Object Model, (DOM) of the page. The element most commonly used for function calls is the <TD> tag element. The Id of <TD> the tag is made up of several distinct components that are interpreted by the JavaScript in order to identify the position of the <TD> tag or cell within the DOM. The following Id is an example of a typical <TD> tag Id representing a cell:

```
<td id='UltraWebGridlrc_3_2_4'>
```

The components of the Id are delimited by the underscore character. The first component indicates the Id of the owning grid. The next component is the object type identifier: in this case 'rc', indicating that the object is a cell. If the object is a cell, the last component indicates the column number within the band that the cell belongs to. The components between the object type identifier and the last position indicate the row position of the cell within the band hierarchy. In this case the Id indicates the 4th column within the 2nd child row of the 3rd top-level row of the grid. The number of bands references by the Id is equal to the total number of components in the Id minus 2.

The following Id is an example of a typical <TH> tag Id representing a column:

```
<td id='UltraWebGridlc_2_4'>
```

The first component indicates the Id of the owning grid. The next component is the object type identifier: in this case 'c', indicating that the object is a column. The next component is the index of the Band to which the column belongs. And the last component is the index of the column within the band.

Functions

To see a description and example of the function, **click on the text of the function.**

If you want to see all of the function definitions and examples expanded, [click here](#).

To collapse all definitions and examples, [click here](#).

function `igcmbo_getComboById(string combold)`

Returns a WebCombo object based on the passed in Id value as it exists on the server. This function is useful in situations where the first parameter of a WebCombo client-side event is being used to identify which control fired the event. It is also necessary for obtaining the oCombo reference from within the **InitializeCombo** event, because the oWebCombo variable has not yet been assigned at that point.

```
var oCombo = igcmbo_getComboById('WebCombo1');
```

function `igtbl_getElementById(string tagId)`

Returns a DOM (Document Object Model) Element object based on its tag Id in a browser independent way. In order to return a valid DOM element, the Id parameter must exist within the current document as the Id attribute of an HTML element.

```
var elem = igtbl_getElementById('MyElemId');  
elem.style.backgroundColor = "Red";
```

function igtbl_getGridById(string gridId)

Returns a Grid object based on its tag Id. The Grid object returned is a JavaScript object that contains many of the properties of UltraWebGrid as it exists within the HTML page. The Id of the Grid object is the name of the Grid on the Server. See Grid Object.

```
var grid = igtbl_getGridById('UltraWebGrid1');  
grid.AllowUpdate = false;
```

function igtbl_getBandById(string cellId)

Returns a Band object based on the tag Id of a Cell within the grid. The function returns the Band object to which the Cell belongs. The Cell Id is the Id tag of the <TD> element representing a Grid Cell within the HTML page. The Band object returned is a JavaScript object that contains the properties of the Band object as it exists within the HTML page. See Band Object.

```
var band = igtbl_getBandById('UltraWebGrid1_2_3_1_4');
```

function igtbl_getColumnById(string cellId)

Returns a Column object based on the tag Id of a Cell within the grid. The function returns the Column object to which the Cell belongs. The CellId parameter is the Id tag of the <TD> element representing a Grid Cell within the HTML page. The Column object returned is a JavaScript object that contains the properties of the Column object as it exists within the HTML page. See Column Object.

```
var column = igtbl_getColumnById('UltraWebGrid1_2_3_1_4');
```

function igtbl_getRowById(string CellId)

Returns a Row object based on the tag Id of a Cell within the grid. The function returns the Row object to which the Cell belongs. The CellId parameter is the Id tag of the <TD> element representing a Grid Cell within the HTML page. The Row object returned is a JavaScript object that contains the properties of the Row object as it exists within the HTML page. See Row Object.

```
var row = igtbl_getRowById('UltraWebGrid1_2_3_1_4');
```

function igtbl_getCellById(string CellId)

Returns a Cell object based on the tag Id of a Cell within the grid. The function returns the Cell object corresponding to the CellId parameter. The CellId parameter is the Id tag of the <TD> element representing a Grid Cell within the HTML page. The Cell object returned is a JavaScript object that contains the properties of the Cell object as it exists within the HTML page. This utility function differs from the function `igtbl_getElementById` in that it returns a new object that exposes special properties of a grid Cell as opposed to the HTML DOM object that represents the <TD> tag within the HTML page. See Cell Object.

```
var cell = igtbl_getCellById('UltraWebGrid1_2_3_1_4');
```

function igtbl_getActiveCell(string gridName)

Returns a Cell object representing the active cell within the grid.

```
var cell = igtbl_getActiveCell('UltraWebGrid');
```

function igtbl_getActiveRow(string gridName)

Returns a Row object representing the active row within the grid

```
var row = igtbl_getActiveRow('UltraWebGrid');
```

function igtbl_isCell(string itemName)

Returns a boolean value that indicates if the item denotes a Cell within the grid

```
var bCell = igtbl_isCell('UltraWebGridlrc_1_3');
```

function igtbl_isRowLabel(string itemName)

Returns a boolean value that indicates if the item denotes a Row within the grid

```
var bRowLabel = igtbl_isRowLabel('UltraWebGridll_1_3');
```

function igtbl_isColumnHeader(string itemName)

Returns a boolean value that indicates if the item denotes a Column within the grid

```
var bColHeader = igtbl_isColumnHeader('UltraWebGrid1c_1_3');
```

function igtbl_getCollapseImage(string gridName, int bandNo)

Returns the resolved CollapseImage string for a band. If the value for the Band is null, the function returns the Grid default.

```
var image = igtbl_getCollapseImage('UltraWebGrid1', 0);
```

function igtbl_getExpandImage(string gridName, int bandNo)

Returns the resolved ExpandImage string for a band. If the value for the Band is null, the function returns the Grid default.

```
var image = igtbl_getExpandImage('UltraWebGrid1', 0);
```

function igtbl_getCellClickAction(string gridName, int bandNo)

Returns the resolved CellClickAction value for a band. If the value for the Band is 0, the function returns the Grid default.

```
var clickAction = igtbl_getCellClickAction('UltraWebGrid1', 0);
```

function igtbl_getSelectTypeCell(string gridName, int bandNo)

Returns the resolved SelectTypeCell value for a band. If the value for the Band is 0, the function returns the Grid default.

```
var selectType = igtbl_getSelectTypeCell('UltraWebGrid1', 0);
```

function igtbl_getSelectTypeColumn(string gridName, int bandNo)

Returns the resolved SelectTypeColumn value for a band. If the value for the Band is 0, the function returns the Grid default.

```
var selectType = igtbl_SelectTypeColumn('UltraWebGrid1', 0);
```

function igtbl_getSelectTypeRow(string gridName, int bandNo)

Returns the resolved SelectTypeRow value for a band. If the value for the Band is 0, the function returns the Grid default.

```
var selectType = igtbl_getSelectTypeRow('UltraWebGrid1', 0);
```

function igtbl_getHeaderClickAction(string gridName, int bandNo, int columnNo)

Returns the resolved HeaderClickAction value for a column. If the value for the Column is 0, the function checks the Band. If the value for the Band is null, the function returns the Grid default.

```
var clickAction = igtbl_getHeaderClickAction('UltraWebGrid1', 0, 2);
```

function igtbl_getAllowUpdate(string gridName, int bandNo, int columnNo)

Returns the resolved AllowUpdate value for a column. If the value for the Column is 0, the function checks the Band. If the value for the Band is null, the function returns the Grid default.

```
var update = igtbl_getAllowUpdate('UltraWebGrid1', 0, 3);
```

function igtbl_getAllowColSizing(string gridName, int bandNo, int columnNo)

Returns the resolved AllowColSizing value for a column. If the value for the Column is 0, the function checks the Band. If the value for the Band is null, the function returns the Grid default.

```
var sizing = igtbl_getAllowColSizing('UltraWebGrid1', 0, 2);
```

function igtbl_getRowSizing(string gridName, int bandNo, string row)

Returns the resolved AllowRowSizing value for a row. If the value for the Row is 0, the function checks the Band. If the value for the Band is null, the function returns the Grid default.

```
var sizing = igtbl_getRowSizing('UltraWebGrid1', 0, 'UltraWebGrid1r_3');
```

function igtbl_getRowSelectors(string gridName, int bandNo)

Returns the resolved RowSelectors value for a band. If the value for the Band is null, the function returns the Grid default.

1 = Yes

2 = No

```
var selectors = igtbl_getRowSelectors('UltraWebGrid1', 0);
```

function igtbl_getNullText(string gridName, int bandNo, int columnNo)

Returns the resolved NullText value for a cell. If both the Column and Band are null, the function returns the Grid default.

```
var image = igtbl_getNullText('UltraWebGrid1', 0, 5);
```

function igtbl_getEditCellClass(string gridName, int bandNo)

Returns the resolved style class name for editing the cells of a band. If the Band property is null, the function returns the Grid default.

```
var class = igtbl_getEditCellClass('UltraWebGrid1', 0);
```

function igtbl_getFooterClass(string gridName, int bandNo, int columnNo)

Returns the resolved style class name for displaying the footer of a column. If the Column and Band values are null, the function returns the Grid default.

```
var Class = igtbl_getFooterClass('UltraWebGrid1', 0, 4);
```

function igtbl_getGroupByRowClass(string gridName, int bandNo)

Returns the resolved style class name for displaying grouped rows of a band. If the Band property is null, the function returns the Grid default.

```
var class = igtbl_getGroupByRowClass('UltraWebGrid1', 0);
```

function igtbl_getHeadClass(string gridName, int bandNo, int columnNo)

Returns the resolved style class name for displaying the header of a column. If the Column and Band values are null, the function returns the Grid default.

```
var class = igtbl_getHeadClass('UltraWebGrid1', 0);
```

function igtbl_getRowLabelClass(string gridName, int bandNo)

Returns the resolved style class name for displaying row labels of a band. If the Band property is null, the function returns the Grid default.

```
var class = igtbl_getRowLabelClass('UltraWebGrid1', 0);
```

function igtbl_getSelGroupByRowClass(string gridName, int bandNo)

Returns the resolved style class name for selected grouped rows of a band. If the Band property is null, the function returns the Grid default.

```
var class = igtbl_getSelGroupByRowClass('UltraWebGrid1', 0);
```

function igtbl_getSelHeadClass(string gridName, int bandNo, int columnNo)

Returns the resolved style class name for displaying the selected column headers. If the Column and Band values are null, the function returns the Grid default.

```
var class = igtbl_getSelHeadClass('UltraWebGrid1', 0);
```

function igtbl_getSelCellClass(string gridName, int bandNo, int columnNo)

Returns the resolved style class name for displaying the selected cells. If the Column and Band values are null, the function returns the Grid default.

```
var class = igtbl_getSelCellClass('UltraWebGrid1', 0);
```

function igtbl_getExpAreaClass(string gridName, int bandNo)

Returns the resolved style class name for displaying the row expansion area of a band. If the Band property is

null, the function returns the Grid default.

```
var class = igtbl_getExpAreaClass('UltraWebGrid1', 0);
```

function igtbl_toggleRow(string gridName, string srcRow, boolean expand)

Toggles the expanded state of a row between expanded and collapsed.

gn - the grid name

srcRow - the Id of the row to be toggled

expand - if True, the rows is expanded. If False, the row is collapsed

```
igtbl_toggleRow('UltraWebGrid1', 'UltraWebGrid1r_1_2', true);
```

function igtbl_clearSelectionAll(string gridName)

Clears the selection state of the grid so that no cells, columns or rows are selected.

gn - the grid name.

```
igtbl_clearSelectionAll('UltraWebGrid1');
```

function igtbl_selectCell(string gridName, string cellId, boolean selFlag, boolean fireEvent)

Turns selection on or off for the specified cell.

gn - the grid name

cellId - the id of the cell

selFlag - if True, the cell is selected. If False, the cell is unselected

fireEvent - if true, an event is fired to listeners.

```
igtbl_selectCell('UltraWebGrid1', 'UltraWebGrid1_1_2_3', true, false);
```

function igtbl_selectRow(string gridName, string rowId, boolean selFlag, boolean fireEvent)

Turns selection on or off for the specified row.

gn - the grid name

rowId - the id of the row

selFlag - if True, the row is selected. If False, the row is unselected

fireEvent - if True, an event is fired to listeners.

```
igtbl_selectRow('UltraWebGrid1', 'UltraWebGrid1r_1_2', true, false);
```

function igtbl_selectColumn(string gridName, string columnId, boolean selFlag, boolean fireEvent)

Turns selection on or off for the specified column.

gn - the grid name

columnId - the id of the column

selFlag - if True, the column is selected. If False, the column is unselected

fireEvent - if True, an event is fired to listeners.

```
igtbl_selectColumn('UltraWebGrid1', 'UltraWebGrid1c_1_2', true, false);
```

function igtbl_setActiveCell(string gridName, element cell)

Makes the specified cell element the active cell in the grid. This cell will have a border rectangle drawn around its perimeter.

gn - the grid name

cell - the cell element

```
var cell= igtbl_getActiveCell();
```

```
cell = cell.NextSibling;
```

```
if(cell != null)
```

```
    igtbl_setActiveCell('UltraWebGrid1', cell);
```

function igtbl_setActiveRow(string gridName, element row)

Makes the specified row element the active row in the grid. This row will have a border rectangle drawn around its perimeter.

gn - the grid name

row - the row element

```
var row = igtbl_getActiveRow();
```



```
row = row.NextSibling;
if(row != null)
    igtbl_setActiveRow('UltraWebGrid1', row);
```

function igtbl_getInnerText(sourceElement)

Returns the InnerText value of an HTML element in a browser-independant way.

```
var innerText = igtbl_getInnerText(igtbl_getElementById('UltraWeGrid1_1_3_4_1'));
var innerText = igtbl_getInnerText('UltraWebGrid1_1_1');
```

function igtbl_setInnerText(sourceElement, strText)

Sets the InnerText value of an HTML element in a browser independant way.

```
igtbl_setInnerText(igtbl_getElementById('UltraWeGrid1_1_3_4_1'), 'Cell Value');
igtbl_setInnerText('UltraWebGrid1_1_2', "newValue");
```

function igtbl_scrollToView(gn, child)

Scrolls a child object of the grid (row or cell) into view.

gn - the grid name

child - the DOM object you want to place into view. The *child* parameter has to be a child of the grid.

```
var row = igtbl_getElementById(rowId);
igtbl_scrollToView(gn, row);
```

No Bands Collection Object

The Bands collection present on the server-side is not replicated as an object on the client-side. Instead, it is implemented on the client-side as an array of values accessible through the **Bands** property of the Grid object. Please consult the [Grid object](#) topic for more information on the **Bands** property.

The Bands property is shown in the Client-Side Object Tree for purposes of clarity, to illustrate how objects are grouped logically in a manner comparable to the server-side object tree.

Client-Side Object Model - Band Object

The Band object on the client represents the styles and behaviors that apply to all rows at a particular level of the grid hierarchy.

The Band Object of UltraWebGrid can be obtained within the page using the `igtbl_getGridById` or the `igtbl_getBandById` utility functions.

```
var band = igtbl_getGridById('UltraWebGrid1').Bands[n];
```

...where *n* is the number of the band you want to access. The top-level band in the grid is Band 0.

```
var band = igtbl_getBandById('cellId');
```

...where *cellId* is the ID of a Cell object contained in the band.

This topic also contains the following sections:

- [Methods](#)

Properties

AllowAddNew

Gets or sets an integer value that determines if new rows can be added to the band.

0 = Not Set

1 = Yes

2 = No

AllowColumnMoving

Gets or sets an integer value that determines if columns can be moved within the band. This property is read-only and should not be changed.

0 = Not Set

1 = Yes

2 = No

AllowColSizing

Gets or sets an integer value that determines if columns can be resized in the band.

0 = Not Set

1 = Fixed

2 = Free

AllowDelete

Gets or sets an integer value that determines if rows can be deleted in the band.

0 = Not Set

1 = Yes

2 = No

AllowSort

Gets or sets an integer value that determines if sorting is supported for the band.

0 = Not Set

1 = Yes

2 = No

AllowUpdate

Gets or sets an integer value that determines if cells in the band can be updated using cell editing. This property is read-only and should not be changed.

0 = Not Set
1 = Yes
2 = No
3 = RowTemplateOnly

AltClass

Gets or sets a CSS class name value that specifies the style for alternate cells in the band. This property can be changed on the client.

```
var band = igtbl_getBandById('cellId');  
band.AltClass = "MyAlternateCssClass";
```

CellClickAction

Gets or sets an integer value that indicates what action will be taken when a cells are clicked in the grid by default. The property can be changed on the client.

0 = NotSet
1 = Edit
2 = RowSelect
3 = CellSelect

ColFootersVisible

Gets an integer value that indicates if column footers are visible in the grid. This property is read-only and should not be changed.

0 = NotSet
1 = Yes
2 = No

ColHeadersVisible

Gets an integer value that indicates if column headers are visible in the grid. This property is read-only and should not be changed.

0 = NotSet
1 = Yes
2 = No

CollapseImage

Gets or sets a string value that determines the image displayed to denote the collapsing of an expanded row. This property can be changed on the client. New image will appear on the newly added or recently expanded rows only.

```
var band = igtbl_getBandById('cellId');  
band.CollapseImage = "MyCollapseImage.gif";
```

Columns

Gets the array of Column objects for the band. The length of this array is equal to the number of columns in the band.

```
var band = igtbl_getBandById('cellId');  
band.columns[0].HeaderClickAction = 1;
```

CurrentEditRowImage

Gets or sets a string value that determines the image displayed when a row is the current, active row and it can be modified using its row template by clicking on the image. This property can be changed on the client. New image will appear after changing the current row.

```
var band = igtbl_getBandById('cellId');  
band.CurrentEditRowImage = "MyCurrentEditRowImage.gif";
```

CurrentRowImage

Gets or sets a string value that determines the image displayed when a row is the current, active row. This property can be changed on the client. New image will appear after changing the current row.

```
var band = igtbl_getBandById('cellId');
```

```
band.CurrentRowImage = "MyCurrentRowImage.gif";
```

DefaultRowHeight

A string value that specifies the default height for rows in the band. This value can be in web units. If the value is changed it will be applied to the newly added rows only.

```
var grid = igtbl_getGridById('G_UltraWebGrid1');  
grid.bands[0].DefaultRowHeight = "16 pt";
```

EditCellClass

Gets or sets a CSS class name value that specifies the style for cells in the band that are being edited. This property can be changed on the client.

```
var band = igtbl_getBandById('CellId');  
band.EditCellClass = "MyEditCssClass";
```

Expandable

Gets an integer value that indicates if rows are expandable on the page. This property is read-only and should not be changed.

0 = Notset

1 = Yes

2 = No

ExpandImage

Gets or sets a string value that determines the image displayed to denote the expansion of a collapsed. This property can be changed on the client. New image will appear on the newly added or recently collapsed rows only.

```
var band = igtbl_getBandById('CellId');  
band.ExpandImage = "MyExpandImage.gif";
```

ExpAreaClass

Gets or sets a CSS class name value that specifies the style for the row expansion areas of the grid. This property can be changed on the client.

```
var band = igtbl_getBandById('CellId');  
band.ExpAreaClass = "MyExpCssClass";
```

FooterClass

Gets or sets a CSS class name value that specifies the style for footer cells in the grid. This property can be changed on the client.

```
var band = igtbl_getBandById('cellId');  
band.FooterClass = "MyFooterCssClass";
```

Grid

Returns the Grid object associated with the band. This property is read-only and should not be changed.

GroupByRowClass

Gets or sets a CSS class name value that specifies the style for grouped rows in the band. This property can be changed on the client.

```
var band = igtbl_getBandById('cellId');  
band.GroupByrowClass = "MyGroupByCssClass";
```

HeaderClass

Gets or sets a CSS class name value that specifies the style for column headers in the band. This property can be changed on the client.

```
var band = igtbl_getBandById('CellId');  
band.HeaderClass = "MyHeaderCssClass";
```

HeaderClickAction

Gets or sets an integer value that indicates what action will be taken when a column headers are clicked in the band. The property can be changed on the client.

0 = NotSet
1 = Select
2 = SortSingle
3 = SortMulti

Index

Returns the index value for the band within the Bands collection. This property is read-only and should not be changed.

IsGrouped

Returns a Boolean value that indicates whether or not the band has grouped columns. This property is read-only and should not be changed.

ItemClass

Gets or sets a CSS class name value that specifies default style for cells in the band. This property can be changed on the client.

```
var band = igtbl_getBandById('CellId');  
band.ItemClass = "MyCssClass";
```

Key

Returns the key string value for the band. This property is read-only and should not be changed.

NonSelHeaderClass

Gets or sets a CSS class name value that specifies the style for non-selected column headers in the band. This property can be changed on the client.

```
var band = igtbl_getBandById('CellId');  
band.NonSelHeaderClass = "MyNonSelHeadCssClass";
```

NullText

Returns the string used to indicate null values in the band. This property can be changed on the client. If the value is changed it will be applied to the newly added rows or to changed cells only.

RowLabelClass

Gets or sets a CSS class name value that specifies the style for row selector labels in the band. This property can be changed on the client.

```
var band = igtbl_getBandById('CellId');  
band.RowLabelClass = "MyRowLabelCssClass";
```

RowSelectors

Gets a Boolean value that indicates if rows selectors are visible on the page. This property is read-only and should not be changed on the client.

0 = Not Set
1 = Yes
2 = No

RowSizing

Gets an integer value that indicates if row sizing is allowed for the band. This property can be changed on the client.

0 = Notset
1 = Fixed
2 = Free
var band = igtbl_getBandById('CellId');
band.RowSizing = 2;

RowTemplate

Gets an integer value that determines the HtmlId of the row template being used for the band.

0 = NotSet

1 = Fixed

2 = Free

```
var band = igtbl_getBandById('CellId');
```

```
band.RowSizing = 2;
```

SelCellClass

Gets or sets a CSS class name value that specifies the style for selected cells in the band. This property can be changed on the client.

```
var band = igtbl_getBandById('CellId');
```

```
band.SelCellClass = "MySelCellCssClass";
```

SelectTypeCell

Gets or sets an integer value that indicates the type of cell selection that is in effect for the band. This property can be changed on the client.

0 = NotSet

1 = None

2 = Single

3 = Extended

```
var band = igtbl_getBandById('CellId');
```

```
band.SelectTypeCell = 2;
```

SelectTypeColumn

Gets or sets an integer value that indicates the type of column selection that is in effect for the band. This property can be changed on the client.

0 = NotSet

1 = None

2 = Single

3 = Extended

```
var band = igtbl_getBandById('CellId');
```

```
band.SelectTypeColumn = 2;
```

SelectTypeRow

Gets or sets an integer value that indicates the type of row selection that is in effect for the band. This property can be changed on the client.

0 = NotSet

1 = None

2 = Single

3 = Extended

```
var band = igtbl_getBandById('CellId');
```

```
band.SelectTypeRow = 2;
```

SelGroupByRowClass

Gets or sets a CSS class name value that specifies the style for selected GroupBy rows in the band. This property can be changed on the client.

```
var band = igtbl_getBandById('CellId');
```

```
band.SelGroupbyRowClass = "MySelGroupByCssClass";
```

SelHeadClass

Gets or sets a CSS class name value that specifies the style for selected column headers in the band. This property can be changed on the client.

```
var band = igtbl_getBandById('CellId');
```

```
band.SelHeadClass = "MySelHeadCssClass";
```

SortedColumns

Returns the collection of Column objects that are being used to sort data in the band. This property is read-only and should not be changed.

Visible

Gets a Boolean value that indicates if the band is visible on the page. This property is read-only and should not be changed.

Methods

getCollapseImage

Returns the resolved CollapseImage string for a band. If the value for the Band is null, the function returns the Grid default.

Parameters:

None.

getColumnFromKey(key)

Returns a Column object based on the specified Key value.

Parameters:

key- The Key value of the Column you want to access.

getExpandImage

Returns the resolved ExpandImage string for a band. If the value for the Band is null, the function returns the Grid default.

Parameters:

None.

getRowAltClassName

Returns the resolved style class name for displaying the alternate cells area of a band. If the Band property is null, the function returns the Grid default.

Parameters:

None.

getRowStyleClassName

Returns the resolved style class name for displaying the cells area of a band. If the Band property is null, the function returns the Grid default.

Parameters:

None.

getSelectTypeCell

Returns a resolved value that determines which type of cell selection is in effect for the band. The value will be one of the enumerations that specifies what selection type is actually in effect. This method will not return 0 (NotSet).

Parameters:

None.

getSelectTypeColumn

Returns a resolved value that determines which type of column selection is in effect for the band. The value will be one of the enumerations that specifies what selection type is actually in effect. This method will not return 0 (NotSet).

Parameters:

None.

getSelectTypeRow

Returns a resolved value that determines which type of row selection is in effect for the band. The value will be one of the enumerations that specifies what selection type is actually in effect. This method will not return 0 (NotSet).

Parameters:

None.

Client-Side Object Model - Row Object

Row objects are not automatically created and populated on the client for each row of the grid. Row objects are created dynamically as they are requested. To obtain the row object for a particular Cell, use the utility function `igtbl_getRowById(cellId)`, where the `cellId` is the target of an event occurring on the client.

If you have access to the Rows collection containing the row, you can use `Rows.getRow(n)` to return a row, where `n` is the index of the desired row within the collection.

Another method that returns a Row object is `igtbl_getActiveRow(gridName)`.

The underlying HTML DOM element of a Row object is available on the Row.Element property.

This topic also contains the following sections:

- [Methods](#)

Properties

Band

Returns the Band object that this Row object is associated with. This property is read-only and should not be changed.

Element

Returns the HTML DOM element (<TR> tag) that is associated with the row object. This property is read-only and should not be changed.

Expandable

Returns a Boolean value that indicates whether the Row is capable of being expanded or not. This property is read-only and should not be changed.

Expanded

Returns a Boolean value that indicates whether the Row is initially expanded or not. This property is read-only and should not be changed. To get current state of the row use the `getExpanded` function.

FirstChildRow

Returns the first child Row object of a Row, if one exists. This property is read-only and should not be changed.

FirstRow

Returns the actual HTML element of the row, which can be different in case of groupby row. This property is read-only and should not be changed.

GroupByRow

Returns a Boolean value that indicates if the Row object is displaying a GroupBy row. This property is read-only and should not be changed.

GroupColId

Returns the Id of the column being used to group rows. This value is only returned if the Row object is object is displaying a GroupBy row (**GroupByRow** = true). This property is read-only and should not be changed.

MaskedValue

Returns the value that is being displayed for a GroupBy row. By default, this property returns the name of the column being used to group the rows. This value is only returned if the Row object is object is displaying a GroupBy row (**GroupByRow** = true). This property can be changed on the client.

OwnerCollection

Returns the Rows collection object that contains the current row. This property is read-only and should not be changed.

ParentRow

Returns the Row object from which the current row is descended. This object will be a row from the previous band. For example, if the current row is in band 1, this will be the row's parent in Band 0. If the current row is a Band 0 row, this property returns Null.

Rows

Returns a collection of the child rows of the current row. This collection will contain all the rows in the following band that are related to the current row. For example, if the current row is in Band 0, the collection returned by **Rows** will contain all of the Band 1 rows that are children of the current row. If the current row has no child rows, or the data being displayed is not hierarchical, this property returns Null. This property is read-only and should not be changed.

Value

Returns the value from the column that is being used to group the rows. This value is only returned if the Row object is object is displaying a GroupBy row (**GroupByRow** = true).

Methods

activate

Sets the current row as the active row. The current row will receive the input focus. Note that invoking this method may deactivate the currently active row or cell, which may have implications if the user is currently editing data.

Parameters:
None.

compare(row)

Compares the current row to the specified row based on the position of the columns within the sorted columns collection of the band containing the rows. This method can only be applied to rows within the same band.

Parameters:
row- The Row object to be compared with the current row.

Returns:
1- The current row's value is greater than the specified row's value.
0- The current row's value is equal to the specified row's value.
-1- The current row's value is less than the specified row's value.

deleteRow

Deletes the current row.

Parameters:
None.

editRow

Opens the row's editing template, provided the row is templated and an editing interface has been defined. This method causes the template (a form containing custom editing controls) to be displayed to the user.

Parameters:
None.

endEditRow(saveChanges)

Closes the row's editing template, optionally saving changes made by the user.

Parameters:
saveChanges- A Boolean value that specifies whether to save the changes made by the user using the

controls of the template. If true, the changes made are saved and applied to the row. If false or omitted, the user's changes are discarded.

find

Searches through the row data for a specified string. [Regular expression syntax](#) is supported for advanced searches.

Parameters:

regExp- The regular expression used to search the data. This may be omitted on subsequent calls to the **find** method; the previously specified expression will be used to perform the search.

searchUp- A Boolean value that specifies the direction of the search. If set to **true** the search is performed from bottom to top. If set to **false** (or omitted) the search is performed from top to bottom.

findNext

Searches through the row data for a specified string, starting at the location of the previous find. [Regular expression syntax](#) is supported for advanced searches.

Parameters:

regExp- The regular expression used to search the data. This may be omitted on subsequent calls to the **find** method; the previously specified expression will be used to perform the search.

searchUp- A Boolean value that specifies the direction of the search. If set to **true** the search is performed from the position of the previous find to the top of the data. If set to **false** (or omitted) the search is performed from the position of the previous find to the bottom of the data.

getCell(index)

Returns the Cell object within the row that corresponds to the specified index parameter. The index is zero based.

getCellByColumn(column)

Returns a specified cell from the current row, based on the Column object containing the cell.

Parameters:

column- Column object that contains the cell you want to retrieve.

getCellFromKey(key)

Returns a specified cell from the current row, based on the key value of the column containing the cell.

Parameters:

key- A string value that evaluates to the key value assigned to one of the columns in the band containing the row.

getChildRow(index)

Returns the child Row object of the given Row that corresponds to the specified index parameter. The index is zero based.

getExpanded

Returns the expanded or collapsed state of the row. This method returns true if the row is expanded, false if the row is collapsed. This method always returns false for rows with no children.

Parameters:

None.

getIndex

Returns the index of the current row in the Rows collection that contains it. The owning Rows collection is accessible through the row's **OwnerCollection** property.

getLeft

Returns the left edge coordinate of the current row.

Parameters:

None.

getNextRow

Returns the next Row object (based on its index) in the Rows collection containing the current row. Any hidden rows are skipped.

getNextTabRow(shift, ignoreCollapse)

Returns the Row object of the row that follows the current one in the tab order.

Parameters:

shift- A Boolean value that specifies the direction of the tab order. Setting this parameter to true has the same effect as using shift-tab: it reverses the tab order and the previous row in the tab order will be returned. If set to false, tab order proceeds in the usual direction.

ignoreCollapse- A Boolean value that specifies whether the collapsed/expanded state of the following row in the tab order should be considered. If set to true, the following row in the tab order will be returned regardless of whether it is visible on screen (i.e. it is the child of a collapsed parent row.) If set to false, the tab order will be dependent on which rows are actually visible on screen at the time.

getPrevRow

Returns the previous Row object (based on its index) in the Rows collection containing the current row. Any hidden rows are skipped.

getSelected

Returns a Boolean value that specifies the current selected state of the row.

Parameters:

None.

getTop

Returns the top edge coordinate of the current row.

Parameters:

None.

remove

Removes the current row from the collection that contains it, as determined by the **OwnerCollection** property.

Parameters:

None.

scrollToView

Scrolls the current row into view. If the current row was up past the top of the screen, it becomes the topmost row. If the current row was down past the bottom of the screen, it becomes the bottommost row.

Parameters:

None.

setExpanded(expanded)

Sets the expanded or collapsed state of the row. This method has no effect on rows with no children.

Parameters:

expanded- A Boolean value that specifies the desired state of the row. If true, the row will be expanded. If false, the row will be collapsed.

setSelected(select)

Specifies a selected state for the current row.

Parameters:

select- A Boolean value that specifies whether the row should be selected. If true, the row becomes selected. If false, the row is deselected.

toggleRow

Expands or collapses and expandable (parent) row depending on its current state. Expanded rows will be collapsed by this method, and collapsed rows will be expanded.

Parameters:

None.

No Columns Collection Object

The Columns collection present on the server-side is not replicated as an object on the client-side. Instead, it is implemented on the client-side as an array of values accessible through the **Columns** property of the Band object. Please consult the [Band object](#) topic for more information on the **Columns** property.

The Columns property is shown in the Client-Side Object Tree for purposes of clarity, to illustrate how objects are grouped logically in a manner comparable to the server-side object tree.

Client-Side Object Model - Column Object

The Column Object of UltraWebGrid can be obtained within the page using the `igtbl_getGridById` or the `igtbl_getColumnById` utility functions.

```
var col = igtbl_getGridById('UltraWebGrid1').Bands[n].Columns[m];
```

...where *n* is the number of the band containing the column you want to access and *m* is the index of the column within that band's Columns collection. The top-level band in the grid is Band 0.

```
var col = igtbl_getColumnById('cellId');
```

...where *cellId* is the ID of a Cell object contained in the column.

This topic also contains the following sections:

- [Methods](#)

Properties

AllowGroupBy

Gets an integer value that indicates if the column can be grouped. This property is read-only and should not be changed.

0 = Not Set

1 = Yes

2 = No

AllowColResizing

Gets or sets an integer value that determines if the column can be resized.

0 = Not Set

1 = Fixed

2 = Free

AllowUpdate

Gets or sets an integer value that determines if cells in the column can be updated using cell editing.

0 = Not Set

1 = Yes

2 = No

3 = RowTemplateOnly

Band

Gets a reference to the Band object that the column belongs to. This property is read-only and should not be changed.

ButtonClass

Gets or sets a CSS class name value that specifies the style for the column when it is selected. This property can be changed on the client.

```
var column = igtbl_getColumnById('CellId');
```

```
column.ButtonClass = "MyButtonClass";
```

CellButtonDisplay

Gets an integer value that determines how dropdown buttons are displayed in the column. This property is read-only and should not be changed.

0 = OnMouseEnter

1 = Always

CellMultiline

An integer value that determines whether the edit box for the column's cells update will support multiple lines of text. If set to 1, the edit box will accept return characters and transform them to the
 tags when editing is done. If set to 0 or 2, one line edit box is used. This property is read-only and should not be changed.

0 = Not Set
1 = Yes
2 = No

Case

Gets or sets an integer value that determines the display of case inside of cells of the column.

0 = Unchanged
1 = Lower
2 = Upper

DataType

Gets an integer value that represents the data type of the items in the cells of the column. This property is read-only and should not be changed.

2, 3, 16, 20 = Unsigned Integer
4 = Float
5 = Double
7 = DateTime
8 = String
11 = Boolean
14 = Currency
17, 18, 19, 21 = Signed Integer

DefaultValue

Gets or sets a string value that determines the default value that will be displayed and stored for the column when the column's data has not yet been supplied.

FieldLength

Gets or sets an integer value that specifies the maximum length for the input strings of the column.

HeaderClickAction

Gets or sets an integer value that indicates what action will be taken when the column header is clicked. The property can be changed on the client.

0 = Not Set
1 = Select
2 = SortSingle
3 = SortMulti

HeaderText

Gets a string value that specifies the text that will be displayed in the column's header. This property is read-only and should not be changed.

Hidden

Gets a Boolean value that specifies whether the column will be visible. This property is read-only and should not be changed. Use the setHidden function to hide/show the column.

Id

Returns the HTML ID of the column on the page. This property is read-only and should not be changed.

Index

Returns the index of the column in the Columns collection. This property is read-only and should not be changed.

IsGroupBy

Returns a Boolean value that indicates whether the column is grouped.

Key

Returns the key string value for the column. This property is read-only and should not be changed.

MaskDisplay

Gets or sets a string value that supplies an input mask format for data entered in editmode.

NullText

Returns the string used to indicate null values in the column. This property can be changed on the client. New text will appear only after modifying a cell or after adding a new row.

SelCellClass

Gets or sets a CSS class name value that specifies the style for selected cells in the column. This property can be changed on the client.

```
var column = igtbl_getColumnById('CellId');  
column.SelCellClass = "MySelCellCssClass";
```

Selected

Gets a Boolean value indicating whether column is currently selected. This property is read-only and should not be changed.

SelHeadClass

Gets or sets a CSS class name value that specifies the style for column header when it is selected. This property can be changed on the client.

```
var column = igtbl_getColumnById('CellId');  
column.SetHeadClass = "MySelHeadCssClass";
```

SortIndicator

Gets an integer value that specifies the type of sorting that is in effect for the column. This property is read-only and should not be changed.

0 = None
1 = Ascending
2 = Descending
3 = Disabled

TemplatedColumn

Gets a Boolean value that specifies whether the column is templated. This property is read-only and should not be changed.

Type

Gets an integer value that indicates the type of display for the column.

0 = Default
3 = Checkbox
5 = Dropdown
7 = Button
9 = Hyperlink

ValueList

Gets a two dimensional array of values that define the options in a valuelist dropdown. The first dimension contains the list of items in the valuelist. Each item is made up of 2 entries: The first entry is the value of the option. The second entry is the display text for the entry.

ValueListClass

Gets or sets a CSS class name value that specifies the style for valuelists in the column. This property can be changed on the client.

```
var column = igtbl_getColumnById('CellId');  
column.SetValueListClass = "MyValueListCssClass";
```

ValueListPrompt

Returns or sets a string value that will be displayed in a column when a dropdown value list is available to populate the column.

Methods

compareCells(*cell1*, *cell2*)

Compares two cells based on the values of those cells. Cells compared must be in the current column. The function is available only if client sorting is allowed.

Parameters:

cell1- The first cell to be compared.

cell2- The second cell to be compared.

Returns:

The return value is based on the setting of **SortIndicator**, which is used to determine the direction of the sort.

	SortIndicator = 1	SortIndicator = 2
Cell 1 > Cell 2	Returns 1	Returns - 1
Cell 1 < Cell 2	Returns - 1	Returns 1
Cell 1 = Cell 2	Returns 0	Returns 0

compareRows(*row1*, *row2*)

Compares two rows based on the values in the current column for those rows. The function is available only if client sorting is allowed.

Parameters:

row1- The first row to be compared.

row2- The second row to be compared.

Returns:

The return value is based on the setting of **SortIndicator**, which is used to determine the direction of the sort.

	SortIndicator = 1	SortIndicator = 2
Row 1 > Row 2	Returns 1	Returns - 1
Row 1 < Row 2	Returns - 1	Returns 1
Row 1 = Row 2	Returns 0	Returns 0

find(*regExp*, *searchUp*)

Searches through the column data for a specified string. [Regular expression syntax](#) is supported for advanced searches.

Parameters:

regExp- The regular expression used to search the data. This may be omitted on subsequent calls to the **find** method; the previously specified expression will be used to perform the search.

searchUp- A Boolean value that specifies the direction of the search. If set to **true** the search is performed from bottom to top. If set to **false** (or omitted) the search is performed from top to bottom.

findNext(*regExp*, *searchUp*)

Searches through the column data for a specified string, starting at the location of the previous find. [Regular expression syntax](#) is supported for advanced searches.

Parameters:

regExp- The regular expression used to search the data. This may be omitted on subsequent calls to the **find** method; the previously specified expression will be used to perform the search.

searchUp- A Boolean value that specifies the direction of the search. If set to **true** the search is performed from the position of the previous find to the top of the data. If set to **false** (or omitted) the search is performed from the position of the previous find to the bottom of the data.

getAllowUpdate

Returns the resolved setting of AllowUpdate that applies to the column. This value always indicates the actual status of whether or not updates are allowed for the column.

Parameters:

None.

getHidden

Returns the current setting of the **Hidden** property, specifying whether or not a column is visible.

Parameters:

None.

setHidden(*hide*)

Sets the **Hidden** property of the column. If **Hidden** is set to true with this method, the column is removed from view on the client.

Parameters:

hide- A Boolean value that specifies whether the column should be hidden or unhidden. If true, the column is removed from view. If false, the column is displayed.

No ValueList Object

The ValueList object present on the server-side is not replicated as an object on the client-side. Instead, it is implemented on the client-side as an array of values accessible through the **ValueLists** property of the Column object. Please consult the [Column object topic](#) for more information on the **ValueList** property and related properties.

The ValueLists property is shown in the Client-Side Object Tree for purposes of clarity, to illustrate how objects are grouped logically in a manner comparable to the server-side object tree.

Client-Side Object Model - Rows Collection

To obtain the collection of top-level (band 0) rows for the grid, use the utility function `igtbl_getGridById(gridId).Rows`, where the *gridId* is the ID of the grid you want to work with.

Each expandable row in the grid also contains its own Rows collection. The collection contains all of the child rows in the following band that are related to the expandable row.

This topic also contains the following sections:

- [Methods](#)

Properties

Band

Returns the Band object that this Rows collection is associated with. This property is read-only and should not be changed.

Element

Returns the HTML DOM element (page) that is associated with the rows collection. This property is read-only and should not be changed.

Grid

Returns the Grid control that this Rows collection is associated with. This property is read-only and should not be changed.

length

Returns the number of rows in the collection. This property is read-only and should not be changed.

ParentRow

Returns the row from which the rows in the collection are descended. This will always be a row in the previous band in the hierarchy. For example, a Rows collection in Band 1 will return a row in Band 0 via its **ParentRow** property. For the Rows collection in Band 0, this property returns Null. This property is read-only and should not be changed.

Methods

getRow(*i*)

Returns the row object at the specified index within the collection. To minimize overhead, the rows collection is created in a lazy fashion. Client-Side Row objects are not created until needed. This method will create a row object for an existing row in the grid, if that javascript object does not already exist. If a row does not exist in the grid at the specified index, this method will return null.

Parameters:

i- index of the row to be returned.

getRowById(*Id*)

Returns the row object with the specified HTML DOM ID.

Parameters:

Id- HTML ID of the row.

indexOf(row)

Returns the index of the specified row within the collection.

Parameters:

row- The row object in the collection whose index you want to determine.

insert(row, i)

Inserts a Row object into the collection at the specified index. Once the row has been inserted, its presence is reflected in the HTML code of the page and the appearance of the grid is altered to include the new row. The **insert** method is used during sorting to re-arrange the order of the rows in the collection.

Parameters:

row- The row object that will be inserted into the collection.

i- The index in the collection where the row will be inserted.

remove(i)

Removes a Row object from the collection at the specified index. Once the row has been removed, its absence is reflected in the HTML code of the page and the appearance of the grid is altered to exclude the deleted row. The **remove** method is used during sorting to re-arrange the order of the rows in the collection.

Parameters:

i- The index in the collection where the row will be removed.

sort

Sorts the collection based on the sorted columns collection of the band containing the collection. This method simply performs a standard sort operation.

Parameters:

None.

Client-Side Object Model - Cell Object

To obtain a cell object, first determine which Row object contains the cell. (If you have access to the Rows collection containing the row, you can use `Rows.getRow(n)` to return a row, where *n* is the index of the desired row within the collection.) Then use the method `Row.getCell(n)` where *n* is the index of the column containing the cell.

This topic also contains the following sections:

- [Methods](#)

Properties

Band

Returns the Band object that this Cell object is associated with. This property is read-only and should not be changed.

Column

Returns the Column object that this Cell object is associated with. This property is read-only and should not be changed.

Element

Returns the HTML DOM element (<TR> tag) that is associated with the row object. This property is read-only and should not be changed.

Index

The index of the cell in the collection of cells that make up the row.

MaskedValue

Returns the value of the cell with the mask applied, if any.

NextSibling

Returns a HTML element representing the next sibling cell within the row. If there are no more siblings, the property returns null. This property is obsolete. Use the `getNextCell` method instead.

Row

Returns the Row object to which the cell belongs. This property is read-only and should not be changed.

PrevSibling

Returns a HTML element representing the previous sibling cell within the row. If the first cell has been reached, the property returns null. This property is obsolete. Use the `getPrevCell` method instead.

Methods

activate

Sets the current cell as the active cell. The current cell will receive the input focus. Note that invoking this method may deactivate the currently active row or cell, which may have implications if the user is currently editing data.

Parameters:
None.

beginEdit

Causes the cell to enter edit mode.

Parameters:
None.

endEdit

Causes the cell to exit edit mode.

Parameters:
None.

getNextCell

Returns the next Cell object (based on its index) in the Cells collection of the current row. Any hidden cells are skipped.

Parameters:
None.

getNextTabCell(shift)

Returns the Cell object of the cell that follows the current one in the tab order.

Parameters:
shift- A Boolean value that specifies the direction of the tab order. Setting this parameter to true has the same effect as using shift-tab: it reverses the tab order and the previous cell in the tab order will be returned. If set to false, tab order proceeds in the usual direction.

getPrevCell

Returns the previous Cell object (based on its index) in the Cells collection of the current row. Any hidden cells are skipped.

Parameters:
None.

getRow

Returns the Row object that contains the current cell.

Parameters:
None.

getValue

Returns the contents of the Cell

Parameters:
None.

scrollToView

Scrolls the current cell into view.

Parameters:
None.

setValue

Sets the value of the cell to the specified text.

Parameters:
None.

Client-Side Object Model - ExpandEffects Object

The properties of this object correspond to the properties of the ExpandEffects Object as it exists within the UltraWebGrid server element. The ExpandEffects object on is used to work with the filter and transition effects functionality provided by Internet Explorer and the element. Internet Explorer 5.0 or greater is required (5.5 or greater recommended.) Some features may be operating-system dependent.

The object becomes available within the band object when row templates are used. The Band Object of UltraWebGrid can be obtained within the page using the `igtbl_getGridById` utility function.

```
var expandfx = igtbl_getGridById('UltraWebGrid1').Bands[n].ExpandEffects;
```

...where *n* is the number of the band you want to access. The top-level band in the grid is Band 0.

```
var expandfx = igtbl_getBandById('cellId').ExpandEffects;
```

...where *cellId* is the ID of a Cell object contained in the band.

Properties

Duration

How long the expand effect lasts, in milliseconds.

```
var band = igtbl_getGridById('UltraWebGrid1').Bands[0];  
band.ExpandEffects.Duration = 100;
```

Opacity

The opacity of the band, or how much of the grid's background is visible underneath.

```
var band = igtbl_getGridById('UltraWebGrid1').Bands[0];  
band.ExpandEffects.Opacity = 50;
```

Type

The style type of the expand effect. Values include:

- 0 = NotSet
- 1 = Fade
- 2 = Iris
- 3 = Slide
- 4 = RandomDissolve
- 5 = GradientWipe
- 6 = Pixelate

ShadowColor

The color value for the shadow effect.

```
var band = igtbl_getGridById('UltraWebGrid1').Bands[0];  
band.ExpandEffects.ShadowColor = "#000044";
```

ShadowWidth

The width of the shadow effect in pixels.

```
var band = igtbl_getGridById('UltraWebGrid1').Bands[n];  
band.ExpandEffects.ShadowWidth = 3;
```

Delay

The amount of time used to delay the opening of submenus in milliseconds.

```
var band = igtbl_getGridById('UltraWebGrid1').Bands[n];  
band.ExpandEffects.Delay = 50;
```

Solve Common Problems (Task-Based Help)

While the purpose of Tutorials is to teach you how to use the product through exercises that illustrate the control's features, the Task-Based Help aims to aid you in quickly accomplishing a specific task. Use Task-Based Help when you just want to know how to accomplish a particular goal.

Briefly, the Tutorials focus on the product and on showing all that it can do. The Task-Based Help focuses on you and on what you want to accomplish right now.

To use Task-Based Help, first think about what it is you are trying to do, then choose whether you want to accomplish the task using server-side or client-side functionality. Select the appropriate sub-section. Next, decide which of the available categories most likely covers the type of activity you are interested in, and expand that category. Finally, select the topic that most closely matches your interest. If your intended activity seems to fit equally well into multiple categories, try any of them; some topics are listed multiple times under different categories.

Bind The Grid To a DataSet

The UltraWebGrid can use any DataSource that implements the IList, IListed or IBindingList Interface as well as the DataTable and DataSet. The most powerful of these is the DataSet, which can contain multiple DataTables with relation constraints to define the hierarchical relationships between the DataTables. To display hierarchical data, the UltraWebGrid must be provided with a DataSource containing two or more DataTables and the DataTable relationships. The DataSet provides the capabilities needed by UltraWebGrid for the display and maintenance of hierarchical data.

This project creates a "Customers" DataTable and an "Orders" DataTable and adds them to a DataSet. Then a relationship between the two DataTables is created and added to the DataSet. Finally, the completed DataSet is bound to the UltraWebGrid.

To obtain the Visual Studio.NET project files associated with this topic, [click this link](#). (This link points to a resource on the Infragistics website and requires an active Internet connection.)

The project consists of the following Classes:

clsCustomerData

The clsCustomerData Class provides a method for the programmatic generation of the Customers DataTable. This code is not reviewed.

clsOrderData

The clsOrderData Class provides a method for the programmatic generation of the Orders DataTable. This code is not reviewed.

WebForm1

The WebForm1 Class contains a **Click** event procedure for the "Bind DataSet to UltraWebGrid" button. This event contains the code relevant to this project.

1. Declare a New DataSet and name it "CustomerOrders":

In Visual Basic:

```
' declare DataSet to contain Hierarchical data
Dim dataset As System.Data.DataSet = New System.Data.DataSet("CustomerOrders")
```

2. Create a DataTable containing Customer data and add it to the DataSet:

In Visual Basic:

```
' make Customers DataTable
Dim custData As New clsCustomerData()
dataset.Tables.Add(custData.MakeDataTable)
```

3. Create a DataTable containing Order data and add it to the DataSet:

In Visual Basic:

```
' make Orders DataTable
Dim ordData As New clsOrderData()
dataset.Tables.Add(ordData.MakeDataTable(dataset.Tables("Customers")))
```

4. Create a relationship between the Customers and Orders and add it to the DataSet:

In Visual Basic:

```
' create customers/orders relationship and add to DataSet
Dim relCustOrder As New DataRelation("CustOrder", dataset.Tables("Customers").Columns
("CustomerID"), dataset.Tables("Orders").Columns("CustomerID"))
dataset.Relations.Add(relCustOrder)

' set the ultrawebgrid's view type to Hierarchical
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.
Hierarchical
```

5. Bind the DataSet to the UltraWebGrid:

In Visual Basic:

```
' bind the DataSet to the Grid
UltraWebGrid1.DataSource = dataset
UltraWebGrid1.DataBind()
```

Bind The Grid To a Data Table

The UltraWebGrid can use any DataSource that implements the IList, IListed or IBindingList Interface as well as the DataTable and DataSet.

The DataTable can contain the results from a single DataBase Query and consists of rows and columns (This is sometimes referred to as Flat data as opposed to Hierarchical data which consists of more than one table and relations).

Many times the UltraWebGrid needs to display only the data from a single DataBase Query or only needs to display rows and columns. The DataTable provides an ideal container for Row/Column data.

To obtain the Visual Studio.NET project files associated with this topic, [click this link](#). (This link points to a resource on the Infragistics website and requires an active Internet connection.)

This project is very simple. A DataTable is created and bound to the UltraWebGrid. All of the code relevant to this project is contained in the **ButtonClick** event for the "Bind DataTable to UltraWebGrid" button. It consists of the code required to create a simple DataTable and bind it to the UltraWebGrid.

1. Declare a New DataTable with the name "Customers":

In Visual Basic:

```
' declare a DataTable to contain the program generated data
Dim dataTable As New System.Data.DataTable("Customers")
```

2. Create a New Column named "CustomerID" of type Int32 and add it to the DataTable:

In Visual Basic:

```
' create and add a CustomerID column
Dim colWork As New DataColumn("CustomerID", GetType(Int32))
dataTable.Columns.Add(colWork)
```

3. Create an array of DataColumnns to contain the columns of the primary key. Add the "CustomerID" column to the array and bind the array to the DataTable PrimaryKey property:

In Visual Basic:

```
' add CustomerID column to key array and bind to DataTable
Dim Keys(0) As DataColumn
Keys(0) = colWork
dataTable.PrimaryKey = Keys
```

4. Create a New Column named "CustomerName" of type String with a MaxLength of 50 characters and add it to the DataTable:

In Visual Basic:

```
' create and add a CustomerName column
colWork = New DataColumn("CustomerName", GetType(String))
colWork.MaxLength = 50
dataTable.Columns.Add(colWork)
```

5. Create a New Column named "LastOrderDate" of type Date and add it to the DataTable:

In Visual Basic:

```
' create and add a LastOrderDate column
colWork = New DataColumn("LastOrderDate", GetType(Date))
dataTable.Columns.Add(colWork)
```

6. Create a new Row, populate the column values and add it to the DataTable:

In Visual Basic:

```
' add a row
Dim row As DataRow = dataTable.NewRow()
row("CustomerID") = 1
row("CustomerName") = "John's Widgets"
row("LastOrderDate") = Now
dataTable.Rows.Add(row)
```

7. Create another new Row, populate the column values and add it to the DataTable:

In Visual Basic:

```
' add another row
row = dataTable.NewRow()
row("CustomerID") = 2
row("CustomerName") = "Fred's Thingamagigs"
row("LastOrderDate") = Now.AddDays(-101)
dataTable.Rows.Add(row)
```

8. Bind the DataTable to the DataSource Property of the UltraWebGrid:

In Visual Basic:

```
' bind DataTable to UltraWebGrid
UltraWebGrid1.DataSource = dataTable
UltraWebGrid1.DataBind()
```

This project is very straightforward and demonstrates the ease with which a DataTable can be created, populated and bound to the UltraWebGrid with very few lines of code.

Bind the Grid to XML Data

This article concerns binding the UltraWebGrid to data residing within an XML file. This article and sample are not intended to be tutorials on developing XML. For information on how to write XML, refer to the [XML Schema Examples topic](#) in the *.NET Framework General Reference* documentation.

Data can be easily brought from an XML file into a DataSet by using the **ReadXML** method. Use this method to fill your dataset. Then set the **DataSource** of the WebGrid to this DataSet and call **DataBind**.

In Visual Basic:

```
Dim dsWebGrid As New DataSet("WebGrid")
Dim filePath As String
filePath = "c:\inetpub\wwwroot\WebGridHelp\WebGridHelp.xml"
dsWebGrid.ReadXml(filePath)

Me.UltraWebGrid1.DataSource = dsWebGrid
Me.UltraWebGrid1.DataBind()
```

In C#:

```
System.Data.DataSet dsWebGrid = new DataSet("WebGrid");
string filePath= "c:\\inetpub\\wwwroot\\WebGridCSQL\\Sample.xml";
dsWebGrid.ReadXml(filePath);

this.UltraWebGrid1.DataSource = dsWebGrid;
this.UltraWebGrid1.DataBind();
```


Bind The Grid To An Array of Classes

The UltraWebGrid can be bound to a custom data class. Any item that you want the grid to be able to display must be a property of the class. Create an array of these objects in an array list, set the datasource of the grid, call databind and the grid will automatically create a column for each property.

The following code illustrates this concept.

In C#:

```
namespace WebGridClassC
{
    /// <summary>
    /// Summary description for WebForm1.
    /// </summary>
    public class WebForm1 : System.Web.UI.Page
    {
        protected Infragistics.WebUI.UltraWebGrid.UltraWebGrid UltraWebGrid1;

        private void Page_Load(object sender, System.EventArgs e)
        {
            // Put user code to initialize the page here
            Stuff s = new Stuff();
            ArrayList al = new ArrayList();
            al.Add(s);

            //works when setting the datasource and calling databind
            this.UltraWebGrid1.DataSource = al;
            this.UltraWebGrid1.DataBind();
        }

        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form Designer.
            //
            InitializeComponent();
            base.OnInit(e);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.Load += new System.EventHandler(this.Page_Load);
        }
        #endregion
    }
}

public class Stuff
{
    private Guid mID = Guid.NewGuid();
    private string mName = "Bill";
    private int mAge = 10;
    private string mAddress = "1199 Birch Lane";
    private string mCity = "Elk River";
}
```

```
private string mState = "MN";
private string mZip = "55330";
private string mCountry = "USA";
private string mHobbies = "Boating, Fishing";
```

```
public Guid ID
{
    get {return mID;}
}
```

```
public string Name
{
    get{return mName;}
}
```

```
public int Age
{
    get{return mAge;}
}
```

```
public string Address
{
    get{return mAddress;}
}
```

```
public string City
{
    get{return mCity;}
}
```

```
public string State
{
    get{return mState;}
}
```

```
public string Zip
{
    get{return mZip;}
}
```

```
public string Country
{
    get {return mCountry;}
}
```

```
public string Hobbies
{
    get {return mHobbies;}
}
```

```
}
}
```

In VB:

```
Public Class WebForm1
    Inherits System.Web.UI.Page

    Dim UltraWebGrid1 As Infragistics.WebUI.UltraWebGrid.UltraWebGrid

    #Region " Web Form Designer Generated Code "
```

```

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()

End Sub

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    'Put user code to initialize the page here
    Dim s As New Stuff()
    Dim al As New ArrayList()
    al.Add(s)

    'works when setting the datasource and calling databind
    Me.UltraWebGrid1.DataSource = al
    Me.UltraWebGrid1.DataBind()

End Sub
End Class

Public class Stuff

    Dim mID As Guid = Guid.NewGuid()
    Dim mName As String = "Bill"
    Dim mAge As Integer = 10
    Dim mAddress as String = "1199 Birch Lane"
    Dim mCity As String = "Elk River"
    Dim mState As String = "MN"
    Dim mZip As String = "55330"
    Dim mCountry As String = "USA"
    Dim mHobbies As String = "Boating, Fishing"

    Public ReadOnly Property ID() As Guid
        Get
            Return mID
        End Get
    End Property

    Public ReadOnly Property Name() As String
        Get
            Return mName
        End Get
    End Property

    Public ReadOnly Property Age() As Integer
        Get
            Return mAge
        End Get
    End Property

    Public ReadOnly Property Address() As String
        Get
            Return mAddress
        End Get

```

```
End Property

Public ReadOnly Property City() As String
    Get
        Return mCity
    End Get
End Property

Public ReadOnly Property State() As String
    Get
        Return mState
    End Get
End Property

Public ReadOnly Property Zip() As String
    Get
        Return mZip
    End Get
End Property

Public ReadOnly Property Country() As String
    Get
        Return mCountry
    End Get
End Property

Public ReadOnly Property Hobbies() As String
    Get
        Return mHobbies
    End Get
End Property

End Class
```

Object Binding

The UltraWebGrid can be bound to a Collection of objects. This makes it possible to create your own data structure, and then bind the UltraWebGrid to it. For this example we will create a two level object hierarchy. Let's assume we want to display Invoices in the grid. Each Invoice object has four properties; ID, Total, LineItems, and DeletedItems. LineItems and DeletedItems are collections of LineItem objects. The properties of an Invoice object are then used as Columns in the grid with the property name used for the ColumnHeaderText, and it's value displayed in the respective cell. A collection, such as LineItems, will be used as a child band of the current Invoice (Row). You can specify which collection to use for the child band by setting band.ChildBandColumn in the InitializeBand event. A LineItem has one property, ItemNum.

We will start with the LineItem class, since it is the most basic. It consists of one property ItemNum of type int.

In C#:

```
public class LineItem
{
    private int itemNum;
    public LineItem(int itemNum)
    {
        this.itemNum=itemNum;
    }
    public int ItemNum
    {
        get{return itemNum;}
        set{itemNum=value;}
    }
}
```

In VB:

```
Public Class LineItem
    Private _itemNum As Integer
    Public Sub New(ByVal itemNum As Integer)
        Me._itemNum = itemNum
    End Sub

    Public Property ItemNum() As Integer
        Get
            Return _itemNum
        End Get
        Set(ByVal Value As Integer)
            _itemNum = Value
        End Set
    End Property
End Class
```

Now we need to create a collection to hold these items. We could simply put them into an ArrayList or some pre-defined collection, but instead let's create our own collection so that we control addition/removal of LineItem objects.

In C#:

```
public class LineItems:CollectionBase
{
    public LineItems():base()
    {}
    public void Add(LineItem item)
    {
        this.InnerList.Add(item);
    }
    public void Remove(LineItem item)
```

```

    {
        this.List.Remove(item);
    }
}

```

in VB:

```

Public Class LineItems
    Inherits CollectionBase

    Public Sub New()
    End Sub
    Public Function Add(ByVal item As LineItem)
        Me.InnerList.Add(item)
    End Function
    Public Function Remove(ByVal item As LineItem)
        Me.List.Remove(item)
    End Function
End Class

```

We are now ready to create the actual Invoice object itself. The Invoice class will have four properties. An ID of type String, a Total which is stored as a double, and two LineItems collections (LineItems and DeletedItems). The constructor takes two params, ID and Total. It also creates a LineItems and DeletedItems collection. It uses an incremental static int, to simulate an AutoNumber field in Access. This value is then used to create a new LineItem, which is added to the LineItems collection.

In C#:

```

public class Invoice
{
    static int itemnum=0;
    private string id;
    private double total=0.0;
    private LineItems lineItems;
    private LineItems deletedItems;
    public Invoice(string id,double total)
    {
        this.id=id;
        this.total=total;
        this.LineItems=new LineItems();
        this.DeletedItems=new LineItems();
        this.LineItems.Add(new LineItem(itemnum++));
        this.DeletedItems.Add(new LineItem(-1));
    }
    public LineItems LineItems
    {
        get{return lineItems;}
        set{lineItems=value;}
    }
    public LineItems DeletedItems
    {
        get{return deletedItems;}
        set{deletedItems=value;}
    }
    public string ID
    {
        get{return id;}
        set{id=value;}
    }
    public double Total
    {
        get{return total;}
        set{total=value;}
    }
}

```

```
}  
}
```

In VB:

```
Public Class Invoice  
  
    Shared _itemnum As Integer = 0  
    Private _id As String  
    Private _total As Double = 0.0  
    Private _lineItems As LineItems  
    Private _deletedItems As LineItems  
  
    Sub New(ByVal id As String, ByVal total As Double)  
        Me.id = id  
        Me.total = total  
        Me.LineItems = New LineItems()  
        Me.DeletedItems = New LineItems()  
        Me.LineItems.Add(New LineItem(++_itemnum))  
        Me.DeletedItems.Add(New LineItem(-1))  
    End Sub  
  
    Public Property LineItems() As LineItems  
        Get  
            Return _lineItems  
        End Get  
        Set(ByVal Value As LineItems)  
            _lineItems = Value  
        End Set  
    End Property  
  
    Public Property DeletedItems() As LineItems  
        Get  
            Return _deletedItems  
        End Get  
        Set(ByVal Value As LineItems)  
            _deletedItems = Value  
        End Set  
    End Property  
  
    Public Property ID() As String  
        Get  
            Return _id  
        End Get  
        Set(ByVal Value As String)  
            _id = Value  
        End Set  
    End Property  
  
    Public Property Total() As Double  
        Get  
            Return _total  
        End Get  
        Set(ByVal Value As Double)  
            _total = Value  
        End Set  
    End Property  
End Class
```

The last thing we need is a collection to store each Invoice (Row). For this example we are using a fixed size array, but any collection will work.

In C#:

```
Invoice[] orders=new Invoice[]{new Invoice("John Doe",99.99),new Invoice("Jane Doe",65.33)};
```

In VB:

```
Dim orders() As Invoice = {New Invoice("John Doe", 22.0), New Invoice("Jane Doe", 99.99)}
```

We now have an array of Invoices, which we can think of as our Rows for the grid. Since an Invoice object has two collections defined (LineItems and DeletedItems) we must set which should be used for the child band. This can be done in the InitializeBandEvent. First we need to check and make sure the band the event was fired for was the band we want. After checking that, we can set which collection to use. We will use ListItems here, but you can also use DeletedItems.

In C#:

```
private void UltraWebGrid1_InitializeBand(object sender, Infragistics.WebUI.UltraWebGrid.BandEventArgs e)
{
    if(e.Band==UltraWebGrid1.Bands[0])e.Band.ChildBandColumn="ListItems";
}
```

In VB:

```
Private Sub UltraWebGrid1_InitializeBand(ByVal sender As Object, ByVal e As Infragistics.WebUI.UltraWebGrid.BandEventArgs) Handles UltraWebGrid1.InitializeBand
    If e.Band.Equals(Me.UltraWebGrid1.Bands(0)) Then e.Band.ChildBandColumn = "DeletedItems"
End Sub
```

Now we can bind the UltraWebGrid to this array of Invoice objects.

In C#:

```
private void Page_Load(object sender, System.EventArgs e)
{
    this.UltraWebGrid1.DataSource=orders;
    this.UltraWebGrid1.DisplayLayout.ViewType=Infragistics.WebUI.UltraWebGrid.ViewType.Hierarchical;
    this.UltraWebGrid1.DataBind();
}
```

In VB

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Me.UltraWebGrid1.DataSource = orders
    Me.UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.Hierarchical
    Me.UltraWebGrid1.DataBind()
End Sub
```

When this WebForm loads, the UltraWebGrid will have two rows. Each Row will have a child band displaying the ItemNum property, of a ListItem object contained in the DeletedItems collection. It should look something like this:

The screenshot shows a blue-themed UltraWebGrid. The main grid has two rows. Each row has a minus sign icon on the left. The first row has '22' in the 'Total' column and 'John Doe' in the 'ID' column. Below this row is a child band with the title 'ItemNum' and a single cell containing '-1'. The second row has '99.99' in the 'Total' column and 'Jane Doe' in the 'ID' column. Below this row is another child band with the title 'ItemNum' and a single cell containing '-1'.

	Total	ID
-	22	John Doe
ItemNum		
	-1	
-	99.99	Jane Doe
ItemNum		
	-1	

Improve Data Binding Efficiency with the DataBinding Event

To centralize code and add efficiency to an application, use the DataBinding event as the place for setting up and configuring runtime data sources for UltraWebGrid. This allows you to have a single location for data binding code for Page_Load, PageIndexChanged, SortColumn, GroupColumn and UngroupColumn events. Each of these events may require that DataBind() be called, but with this technique, the code to set up the data source does not have to be duplicated.

In Visual Basic:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Me.UltraWebGrid1.DataBind()
End Sub

Private Sub UltraWebGrid1_DataBinding(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles UltraWebGrid1.DataBinding
    Dim conn as System.Data.SqlClient.SqlConnection = New System.Data.SqlClient.SqlConnection
    ("dataSourceString")
    Dim da As System.Data.SqlClient.SqlDataAdapter = New System.Data.SqlClient.SqlDataAdapter
    ("Select * From Customers",conn)
    Dim ds as System.Data.DataSet = New DataSet()
    da.Fill(ds,"Customers")
    Me.UltraWebGrid1.DataSource = ds
    Me.UltraWebGrid1.DataMember = "Customers"
End Sub

Private Sub UltraWebGrid1_PageIndexChanged(ByVal sender As System.Object, ByVal e As Infragistics.WebUI.UltraWebGrid.PageEventArgs ) Handles UltraWebGrid1.PageIndexChanged
    Me.UltraWebGrid1.DisplayLayout.Pager.CurrentPageIndex = e.NewPageIndex
    Me.UltraWebGrid1.DataBind()
End Sub
```

In C#:

```
private void Page_Load(object sender, System.EventArgs e)
{
    this.UltraWebGrid1.DataBind();
}

private void UltraWebGrid1_DataBinding(object sender, System.EventArgs e)
{
    System.Data.SqlClient.SqlConnection cn = new System.Data.SqlClient.SqlConnection
    (dataSourceString);
    System.Data.SqlClient.SqlDataAdapter da = new System.Data.SqlClient.SqlDataAdapter("Select
    * From Customers",cn);
    System.Data.DataSet ds = new DataSet();
    da.Fill(ds,"Customers");
    this.UltraWebGrid1.DataSource = ds;
    this.UltraWebGrid1.DataMember = "Customers";
}

private void UltraWebGrid1_PageIndexChanged(object sender, Infragistics.WebUI.UltraWebGrid.
PageEventArgs e)
{
    this.UltraWebGrid1.DisplayLayout.Pager.CurrentPageIndex = e.NewPageIndex;
    this.UltraWebGrid1.DataBind();
}
```

Filtering Columns From a Data Source

By default all columns of a data source are added to the grid during a databind operation. In order to remove columns in the datasource that are not to appear on the page, set the Hidden property to true.

Columns can also be eliminated from a database bound grid by modifying the select statement for the query or by using a System.Data.DataTable class to construct the exact view of the data that you want.

In Visual Basic (inside of the InitializeLayout event handler):

```
e.Layout.Bands(0).Columns.FromKey("Salary").Hidden = True
```

In C# (inside of the InitializeLayout event handler):

```
e.Layout.Bands[0].Columns.FromKey("Salary").Hidden = True;
```

Bind the Grid to a Hierarchical Data Source

This topic gives you a brief overview of how to place the UltraWebGrid on a form and bind it to a hierarchical data source using the design-time environment of Visual Studio.NET. You will see how to set up the data connection and adapter components, populate the dataset, and display hierarchical data in the grid, all using only two lines of code!

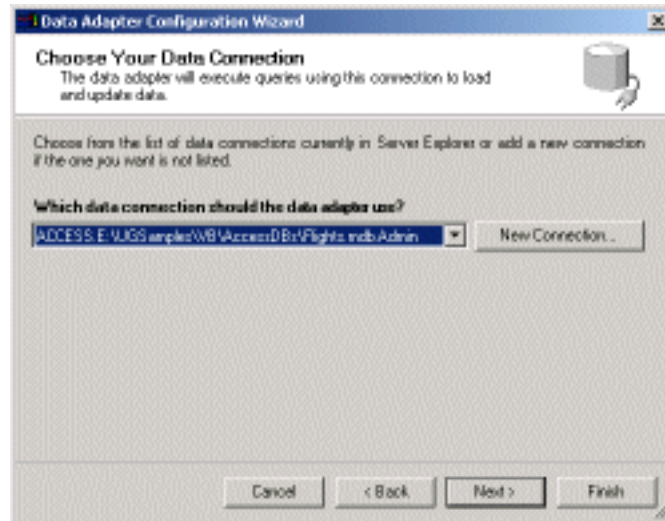
1. Open Visual Studio.NET. You will see the Start screen. (Depending on how you have set up your environment, this screen may not appear.)
2. Create a new project by selecting "New Project."
3. When prompted for the type of project to create, choose Visual Basic or C# ASP.NET Web Application. Supply a name for the application and click OK.
4. The web application project will appear, with a single web form visible. Your toolbox should also be visible. If it is not, display it.

Optional: Before you add the UltraWebGrid to your project, you may want to create a new tab in your toolbox for your Infragistics controls. If so, add the Infragistics tab now and activate it.

Once the tab is displayed that will hold the UltraWebGrid control icon, right-click on the toolbox and choose "Customize Toolbox" from the context menu. The Add Components dialog will appear.

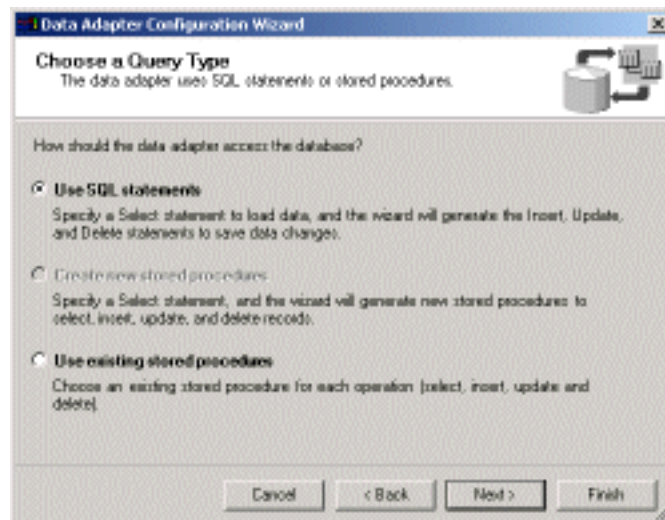
5. Select the ".NET Framework Components" tab.
6. Choose "UltraWebGrid" from the list and put a check mark next to it, then click OK. It will appear on the active tab in the toolbox.
7. Select the UltraWebGrid control in your toolbox and draw the control on the form.
You should note at this point that references to Infragistics.Shared and Infragistics.WebUI.UltraWebGrid have been added to the Solution Explorer window under the "References" section.
8. Click the Data tab in the toolbox. Select the OleDbConnection tool and double-click it or drag it onto the form. This is a non-visual control, so it will appear in the component tray immediately under the form.
9. Click on the OleDbConnection control in the form's component tray and select the **ConnectionString** property in the property sheet. A dropdown list appears that contains the existing data connections, if any. At the bottom of the list is a "New Connection..." option.
10. Select "New Connection..." The first screen of the Connection Wizard will appear.
11. Click the Provider tab and select the "MS Jet 4.0 OLEDB Provider" option from the list. Click the "Next" button. The Connection tab will become active.
12. On the Connection tab, you are prompted to select or enter a database name. Click the ellipsis (...) button to the right of the text box to browse for the database that you will use. Choose the sample database that ships with UltraWebGrid. Navigate to the directory where you installed UltraWebGrid and locate the `NWind.mdb` file in the `samples\data` folder. When you have selected the file, click "Open" and the filename will appear on the Connection tab.
13. Click the "Test Connection" button to verify the connection. Once you see a message indicating that the connection was successful, you can click OK to complete the connection setup.
14. Select OleDbDataAdapter control from your toolbox and double-click it or drag it onto your form. The Data Adapter Configuration Wizard will automatically appear. The first screen of this wizard is a welcome screen; read the text then hit Next to go on to the next step of the wizard.
15. On the second screen of the wizard, you are shown a list of the available database connections and prompted for the database connection to use. Select the one you just created; it will begin with "ACCESS." and include the full path to the database, plus table and login information. So if you have installed UltraWebGrid to a default location, the choice will look something like "ACCESS.C:\Program

Files\Infragistics\UltraWebGrid\v1.00.6007\Samples\Data\NWind.mdb.Admin".



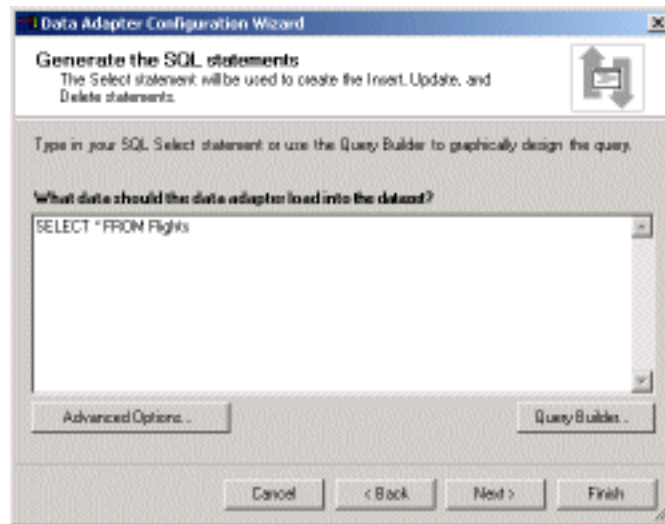
When you have selected the connection, click the "Next" button.

16. On the next screen of the wizard, you are prompted to choose a query type. Choose "Use SQL statements".



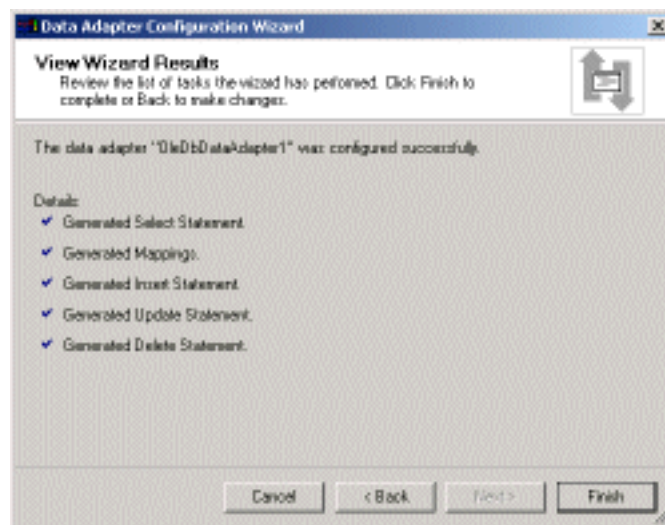
Click the "Next" button to go to the next screen.

17. The following screen prompts you for the SQL statements to generate. You can enter SQL directly into the "What data should the data adapter load into the dataset?" text box. Type "SELECT * FROM Customers" into the text box.



Click the "Next" button when you are done.

18. View the wizard results. All items should be checked on this screen.



When you have verified the wizard settings, click "Finish". The OleDbDataAdapter control will then appear in the form's component tray.

19. Select the OleDbDataAdapter control from your toolbox again and double-click it or drag it onto your form. The Data Adapter Configuration Wizard will appear a second time. Click "Next" to bypass the welcome screen and go on to the next step of the wizard.
20. On the second screen of the wizard, you are shown a list of the available database connections and prompted for the database connection to use. Select the one you previously created (the same one you used for the first adapter); it will begin with "ACCESS." and include the full path to the database. When you have selected the connection, click the "Next" button.
21. On the next screen of the wizard, you are prompted to choose a query type. Choose "Use SQL statements".
Click the "Next" button to go to the next screen.
22. The following screen prompts you for the SQL statements to generate. Type "SELECT * FROM Orders" into the text box.
Click the "Next" button when you are done.
23. View the wizard results. If you receive warnings at this point, you may safely ignore them.
When you have verified the wizard settings, click "Finish". The second OleDbDataAdapter control will then appear in the form's component tray.

24. Select the first OleDbDataAdapter control you created from the form's component tray, go to the property sheet and click the link at the bottom which says "Generate Dataset..." The Generate Dataset screen appears.
25. On the Generate DataSet screen, the top section offers you a "Choose DataSet" option. You should select the "New" option. A default name (DataSet1) is provided. The bottom section of the screen offers a "Choose which table(s) to add to the dataset:" list. You should see both the Customers and Orders tables listed.

Check the boxes next to *both* tables, then click OK. DataSet1 appears in the form's component tray. (You may want to check the name to make sure it is the one you specified.)
26. Select DataSet1 in the form's component tray. In the property sheet, click the "View Schema" link at the bottom of the window. A new tabbed development window will open, displaying the data schema containing the two tables.
27. Right-click on the Customers table. When the context menu appears, select "Add" and then "New Relation..." from the sub-menu. The Edit Relation dialog appears.
28. In the top part of the Edit Relation dialog, note the default name and verify that Customers is "Parent Element" and Orders is "Child Element". In the fields table that appears at the bottom of the dialog, verify that the Key Fields and Foreign Key Fields columns both list "CustomerID." Click OK. The relationship will appear as a dotted line with a diamond that connects the two tables.
29. Switch back to the form containing the UltraWebGrid. Click on the control to select it, then select the **DataSource** property. A drop-down list displays the available options; choose the one that says "DataSet1".
30. Select the **DisplayLayout** property from the property sheet and expand it to reveal the properties that are grouped under it. Locate the **ViewType** property and change its setting to "Hierarchical".
31. In the **Load** event of Form1 enter the following code to fill the Dataset with data from the database and connect the UltraWebGrid to the dataset:

In Visual Basic, enter:

```
OleDbDataAdapter1.Fill(DataSet1)
OleDbDataAdapter2.Fill(DataSet1)
UltraWebGrid1.DataBind()
```

If you are using C#, the syntax would be as follows:

```
OleDbDataAdapter1.Fill(DataSet1, "Customers");
OleDbDataAdapter2.Fill(DataSet1, "Orders");
UltraWebGrid1.DataBind();
```
32. Click Run. The project will execute, and you will see the UltraWebGrid on the form filled with hierarchical data from the Customers and Orders tables in a master/detail relationship. Expand the row for any customer to see a list of the orders for that customer.

Populate ValueListItems From In Memory Dataset

The following code demonstrates how to populate a ValueList with items from an in-memory dataset.

Note that in order to use the ValueList in the WebGrid, you must set the properties of the WebGrid as follows:

```
Column(1).Type = 'DropDownList'  
DisplayLayout.AllowUpdateDefault = 'Yes'  
DisplayLayout.CellClickActionDefault = 'Edit'
```

In Visual Basic:

```
Dim objDTable As New DataTable("Customers")  
objDTable.Columns.Add("First Names")  
Dim objRow1 As System.Data.DataRow  
Dim objRow2 As System.Data.DataRow  
Dim i As Integer  
  
objRow1 = objDTable.NewRow()  
objRow2 = objDTable.NewRow()  
  
objRow1("First Names") = "Steve"  
objRow2("First Names") = "John"  
  
objDTable.Rows.Add(objRow1)  
objDTable.Rows.Add(objRow2)  
  
Dim objValueList As New Infragistics.WebUI.UltraWebGrid.ValueList()  
For i = 0 To objDTable.Rows.Count - 1  
valueList.ValueListItems.Add (objDTable.Rows(i).ItemArray.GetValue(0))  
Next i  
Me.UltraWebGrid1.DisplayLayout.Bands(0).Columns(1).ValueList = valueList
```

In C#:

```
System.Data.DataTable objDTable = new DataTable("Customers");  
objDTable.Columns.Add("First Names");  
  
System.Data.DataRow objRow1;  
System.Data.DataRow objRow2;  
  
objRow1 = objDTable.NewRow();  
objRow2 = objDTable.NewRow();  
  
objRow1["First Names"] = "Steve";  
objRow2["First Names"] = "John";  
objDTable.Rows.Add(objRow1);  
objDTable.Rows.Add(objRow2);  
  
Infragistics.WebUI.UltraWebGrid.ValueList valueList = new Infragistics.WebUI.UltraWebGrid.  
ValueList();  
for(int i = 0; i < objDTable.Rows.Count; i++)  
{  
    valueList.ValueListItems.Add (objDTable.Rows[i].ItemArray.GetValue(0));  
}  
this.UltraWebGrid1.DisplayLayout.Bands[0].Columns[1].ValueList = valueList;
```

Validate Cells on Server-Side

Data validation can be done on the server-side or the client-side. To validate the data on the server-side, use the UpdateCell or UpdateCellBatch events. The UpdateCell event will fire every time a cell in the UltraWebGrid is updated; the UpdateCellBatch will only fire for each updated cell on the next postback. This event is useful for when a submit button is used.

For this task-based help, the UpdateCellBatch event is used. The data for the second column will be checked to ensure that the user is not entering bad data.

The first part of the code checks the index of the column of the cell that is being updated. If the cell is in the second column, processing continues. If "Bad Data" has been entered, then the value of the cell is changed to "Re-enter Data" and the bgcolor of the cell is made red as a visual flag to the user.

In Visual Basic:

```
Private Sub UltraWebGrid1_UpdateCellBatch(ByVal sender As Object, ByVal e As Infragistics.WebUI.UltraWebGrid.CellEventArgs) Handles UltraWebGrid1.UpdateCellBatch
    If e.Cell.Column.Index = 1 Then
        If e.Cell.Value.ToString() = "Bad Data" Then
            TextBox1.Text = "Bad Data has been entered!!"
            e.Cell.Style.BackColor = Color.Red
            e.Cell.Value = "Re-enter Data"
        End If
    End If
End Sub
```

In C#:

```
private void UltraWebGrid1_UpdateCellBatch(object sender, Infragistics.WebUI.UltraWebGrid.CellEventArgs e)
{
    if (e.Cell.Column.Index == 1)
    {
        if (e.Cell.Value.ToString() == "Bad Data")
        {
            TextBox1.Text = "Bad Data has been entered!!";
            e.Cell.Style.BackColor = Color.Red;
            e.Cell.Value = "Re-enter Data";
        }
    }
}
```

If you would like to validate data on the client-side, please reference the article entitled [Prevent Cell From Losing Focus with Invalid Data on Client-Side](#). To validate data as it is loaded into the UltraWebGrid from the datasource, you can use the **InitializeRow** event. For more information on that functionality, reference the topic entitled [Validate Cells on Data Load](#).

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #2**.

Prevent Cell From Losing Focus with Invalid Data on Client-Side

It may prove beneficial to validate user input within the client-side events. The **BeforeExitEditModeHandler** allows for a perfect time to validate the the input and "lock" the user on the cell until proper data has been entered.

First, add a handler for the client-side event, **BeforeExitEditModeHandler**. This property can be set either in the Property Pages under DisplayLayout -> ClientSideEvents -> BeforeExitEditModeHandler or in code:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ClientSideEvents.BeforeExitEditModeHandler = "BeforeExitEditMode"
```

In C#:

```
this.UltraWebGrid1.DisplayLayout.ClientSideEvents.BeforeExitEditModeHandler =  
"BeforeExitEditMode";
```

This event receives two parameters: gridname and cell id. To keep the user within the cell, return true to cancel the event. Here is a JavaScript sample that will not allow a user to enter a date that is after today for a birthdate:

```
//create a global variable so alertbox does not appear twice. box appears twice since in the  
BeforeEndEditMode you are calling alert(message) which tries to end the edit as well  
//without this then you would find yourself in an infinite loop or have an alert appear twice.
```

```
var oldvalue=null;  
function beforeExitEditMode(gridName,cellID)  
{  
    //get the cell  
    cell=igtbl_getCellById(cellID);  
    //get the grid  
    grid=igtbl_getGridById(gridName);  
    //check to see if the user is in the birthdate column  
    if(cell.Column.Key=="BirthDate")  
    {  
        //get today's date  
        var today=new Date();  
        //get the value of the current cell  
        var valDate=new Date(cell.getValue());  
  
        if(valDate < today)  
        {  
            //if the date is before today then allow them to exit edit mode  
            return 0;  
        }  
        //won't get here unless date is greater then today since a valid date would have  
        exited the function above  
        //the next line is necessary so we know that this is the first time the user is  
        coming through the function and  
        //we avoid duplicate alert messages  
        if(oldvalue==null)  
        {  
            //set oldvalue to current value so this is not hit next time through and  
            display alert box  
            oldvalue=valDate;  
            alert("BirthDate can't be in the future");  
        }  
    }  
}
```

```
        //if the user has gotten this far then value is invlaid so return 1 to cancel
the exiting of edit mode
        return 1;
    }
}
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #2**.

Locate Rows Using a Composite Key

This topic describes a general methodology that is not specific to the UltraWebGrid. For more information, Refer to [the DataRowCollection.Find Method topic](#) in the .NET Framework Class Library documentation.

Find a row in the dataset based on the array of values you pass it. The array would contain all the fields that make up your composite key.

The following example uses the values of an array to find a specific row in a collection of DataRow objects. The method presumes a DataTable exists with three primary key columns. After creating an array of the values, the code uses the Find method with the array to get the particular object desired.

Note This example shows how to use one of the overloaded versions of Find. For other examples that might be available, see the individual overload topics.

In Visual Basic:

```
Private Sub FindInMultiPKey(ByVal myTable As DataTable)
    Dim foundRow As DataRow
    ' Create an array for the key values to find.
    Dim findTheseVals(2) As Object
    ' Set the values of the keys to find.
    findTheseVals(0) = "John"
    findTheseVals(1) = "Smith"
    findTheseVals(2) = "5 Main St."
    foundRow = myTable.Rows.Find(findTheseVals)
    ' Display column 1 of the found row.
    If Not (foundRow Is Nothing) Then
        Console.WriteLine(foundRow(1).ToString())
    End If
End Sub
```

In C#:

```
private void FindInMultiPKey(DataTable myTable){
    DataRow foundRow;
    // Create an array for the key values to find.
    object[]findTheseVals = new object[3];
    // Set the values of the keys to find.
    findTheseVals[0] = "John";
    findTheseVals[1] = "Smith";
    findTheseVals[2] = "5 Main St.";
    foundRow = myTable.Rows.Find(findTheseVals);
    // Display column 1 of the found row.
    if(foundRow != null)
        Console.WriteLine(foundRow[1]);
}
```

There is an overloaded find method that can take an array of values, so in the **UpdateCell** event, for example, if you are searching for that row in your dataset, pass the find method an array of values from the cells that make up your primary key (i.e. InvoiceID and InvoiceDetailID from an invoice detail table). It would return that unique row.

Create an Unbound Column

It is possible to add an unbound Column to a grid. This is useful for displaying calculations based on other fields in the row, or for placing check boxes into the grid so users can select multiple rows.

There are two ways to add an unbound column to the grid. At Design-time, you can use the custom property pages. At run-time, you can add a column using the Columns Collection of a Band.

1. Property Pages (Design-time)

- a. Select the grid and click on the **Bands...** collection from the property sheet. Here you can see a list of bands and columns based on the datasource of the grid
- b. Select the **Columns...** collection from the Band you want to add a column to.
- c. Now click the **Add** button at the bottom of the properties page dialog. A new column is created with default HeaderText.
- d. You can change the HeaderText by typing in a new value, like "CalculatedColumn". When you are finished, click **OK**

Code (Run-time)

- a. To add the column in code, access the **Add** method of the **Columns** Collection specifying a key value for the new column. You can optionally specify a caption to be displayed in the column's header.

```
UltraWebGrid1.DisplayLayout.Bands(0).Columns.Add "CalculatedColumn",  
"Calculated Value"
```

2. Once you have the unbound column established, you can use it just like any other column. Most commonly, you will use it for calculations or as a checkbox.

Calculated Column

- a. Assume the grid has two columns, UnitPrice and Quantity, and that you just added an Unbound Column called Total. You would likely populate the Total column using the **InitializeRow** event.

```
e.Row.Cells.FromKey("Total").Value = e.Row.Cells.FromKey("UnitPrice").Value *  
e.Row.Cells.FromKey("Quantity").Value
```

CheckBox

- a. To make the column appear as a check box, set the **Type** property. The best place to do this is in the **InitializeLayout** event. Assuming that the Unbound column you created is in Band 0 and is named "CheckBox" the code would look like this.

```
UltraWebGrid1.DisplayLayout.Bands(0).Columns.FromKey("CheckBox").Type =  
Infragistics.WebUI.UltraWebGrid.ColumnType.CheckBox
```

Set Up GroupBy Mode In Code

The following code demonstrates how to programmatically enable GroupBy mode and then group records by a specified value. In this example, records will appear grouped according to the value of the "CustomerID" field.

In Visual Basic:

```
Me.UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.  
OutlookGroupBy  
Me.UltraWebGrid1.DisplayLayout.Bands(0).Columns.FromKey("CustomerID").IsGroupByColumn=True
```

In C#:

```
this.UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.  
OutlookGroupBy;  
this.UltraWebGrid1.DisplayLayout.Bands[0].Columns.FromKey("CustomerID").  
IsGroupByColumn=true;
```

Handling Sorting Events

To allow the WebGrid to handle sorting you must set the **DisplayLayout.AllowSortingDefault** property. Also, be sure to set the **DisplayLayout.HeaderClickActionDefault** property to give your users the ability to sort the columns by clicking on the column's header. This property can be set to allow sorting on one column at a time or multiple column sorting.

Use this code to allow your users to sort multiple columns when clicking on the headers and holding down the SHIFT key.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.AllowSortingDefault = Infragistics.WebUI.UltraWebGrid.AllowSorting.  
Yes  
UltraWebGrid1.DisplayLayout.HeaderClickActionDefault = Infragistics.WebUI.UltraWebGrid.  
HeaderClickAction.SortSingle
```

In C#:

```
UltraWebGrid1.DisplayLayout.AllowSortingDefault = Infragistics.WebUI.UltraWebGrid.AllowSorting.  
Yes;  
UltraWebGrid1.DisplayLayout.HeaderClickActionDefault = Infragistics.WebUI.UltraWebGrid.  
HeaderClickAction.SortSingle;
```

Custom Sorting

You can also override the grid's sorting functionality and create your own custom sorting rules. First, you must set the **DisplayLayout.EnableInternalRowsManagement** property equal to False. Then be sure to set **e.Cancel** property to True within the **SortColumn** event. Within that event, you will want to place your custom code to sort the column whose header was clicked on. The code we employ here uses the Sort property off of the DefaultView object of the table in our dataset to enable sorting on the column whose header is clicked on. The keyword "DESC" is used to sort this column in descending order (the default would be ascending order). The column index of the clicked header is passed into the event as **e.ColumnNo**. The datasource is reset and the UltraWebGrid rebound to ensure the changes to the dataset are reflected in the grid.

In Visual Basic:

```
e.Cancel = True  
DataSet11.Customers.DefaultView.Sort = UltraWebGrid1.DisplayLayout.Bands(0).Columns(e.  
ColumnNo).Key & " DESC"  
UltraWebGrid1.DataSource = DataSet11  
UltraWebGrid1.DataBind()
```

In C#:

```
e.Cancel = true;  
DataSet11.Customers.DefaultView.Sort = UltraWebGrid1.DisplayLayout.Bands(0).Columns(e.  
ColumnNo).Key + " DESC";  
UltraWebGrid1.DataSource = DataSet11;  
UltraWebGrid1.DataBind();
```

Handling Paging Events

Custom Paging is a powerful way to control the look and feel of paging within UltraWebGrid. When this property is turned on in the Pager object, the application gets control of all information regarding the data to be rendered, the size of the page, and what the labels for the various pages look like in the browser.

Using the CustomLabels property of the Pager object, (available at runtime only), an Enumeration of strings or objects can be provided and whatever string values there are in the collection will be displayed as the paging labels. Normally, a simple array as in the sample below can be provided, while more sophisticated requirements can utilize an enumerable list of any kind to provide the paging label information.

Once the application receives a PageIndexChanged event it can retrieve the index of the page item that was clicked. Using that index, the application decides what page needs to be rendered, fetches the data for that page and calls DataBind on the grid to load the data. The grid will then display the page and the page labels once again.

To implement custom paging for the UltraWebGrid, you need to follow three steps:

1. Set **Pager.AllowCustomPaging** = True.
By setting this property, the UltraWebGrid will ignore settings for Pager.PageSize, allowing you to control the number of rows on each page.
2. Set **EnableInternalRowsManagement** = False.
By setting this property, the UltraWebGrid will only store the rows for the current page. This also forces the UltraWebGrid to fire the PageIndexChanged event, when a user clicks on a new page label.
3. Add code to the **PageIndexChanged** event.
You need to add code to this event to populate the grid with data for the new page. You can get the NewPageIndex from the PageEventArgs object, and use it to determine what data should be displayed.

Here is a code example of a CustomPaging implementation which separates pages alphabetically.

```
In C#:
string[] alphabet=new string[]
{"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"};
private void Page_Load(object sender, System.EventArgs e)
{
    this.UltraWebGrid1.DisplayLayout.Pager.CustomLabels=alphabet;
    if (!this.IsPostBack)
    {
        this.UltraWebGrid1.DisplayLayout.Pager.StyleMode=Infragistics.WebUI.UltraWebGrid.PagerStyleMode.
        CustomLabels;
        this.UltraWebGrid1.DisplayLayout.EnableInternalRowsManagement=false;
        this.UltraWebGrid1.DisplayLayout.Pager.AllowCustomPaging=true;
        doData(); //Assign datasource, call DataBind().
    }
}

private void UltraWebGrid1_PageIndexChanged(object sender, Infragistics.WebUI.UltraWebGrid.
PageEventArgs e)
{
    // Use WHERE to select on the records with CustomerID's that begin with the proper letter.
    OleDbSelectCommand1.CommandText="SELECT CustomerID, ContactName, CompanyName, Phone, Fax, Address
FROM Customers WHERE (CustomerID LIKE '"+alphabet[e.NewPageIndex-1]+"%')";
    // Refill dataset, rebind UltraWebGrid
    doData();
}

private void doData()
{
    OleDbDataAdapter1.Fill(dataSet11);
    UltraWebGrid1.DataBind();
    // Since the grid only has data for the current page, you must specifically set the total number of
    pages.
    this.UltraWebGrid1.DisplayLayout.Pager.PageCount=26;
}
}
```

In Visual Basic:

```
Dim alphabet() As String = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O",  
"P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"}
```

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load  
    Me.UltraWebGrid1.DisplayLayout.Pager.CustomLabels = alphabet  
    If (Not Me.IsPostBack) Then  
        Me.UltraWebGrid1.DisplayLayout.Pager.StyleMode=Infragistics.WebUI.UltraWebGrid.PagerStyleMode.  
        CustomLabels  
        Me.UltraWebGrid1.DisplayLayout.EnableInternalRowsManagement=False  
        Me.UltraWebGrid1.DisplayLayout.Pager.AllowCustomPaging=True  
        doData() 'Assign datasource, call DataBind().  
    End If  
End Sub
```

```
Private Sub UltraWebGrid1_PageIndexChanged(ByVal sender As Object, ByVal e As Infragistics.WebUI.  
UltraWebGrid.PageEventArgs) Handles UltraWebGrid1.PageIndexChanged  
    ' Use WHERE to select on the records with CustomerID's that begin with the proper letter.  
    OleDbSelectCommand1.CommandText = "SELECT CustomerID, ContactName, CompanyName, Phone, Fax, Address  
    FROM Customers WHERE (CustomerID LIKE '" & alphabet(e.NewPageIndex - 1) & "%')"  
    ' Refill dataset, rebind UltraWebGrid  
    doData()  
End Sub
```

```
Private Sub doData()  
    OleDbDataAdapter1.Fill(DataSet1)  
    UltraWebGrid1.DataSource = DataSet1  
    UltraWebGrid1.DataBind() UltraWebGrid1.DataSource = DataSet1  
    ' Since the grid only has data for the current page, you must specifically set the total number of  
    pages.  
    Me.UltraWebGrid1.DisplayLayout.Pager.PageCount = 26  
End Sub
```


Display One Value But Store Another With Value Lists

Using ValueLists you can display one value and store another:

1. Create a ValueList.

```
Dim valueList As Infragistics.WebUI.UltraWebGrid.ValueList  
valueList = new Infragistics.WebUI.UltraWebGrid.ValueList()
```

2. In order to show a particular value and save another value, you must work with the **DataValue** property as well as the **DisplayText** property when you add ValueListItems to the ValueList. You can pass both of these values to the **Add** method of the ValueListItems collection when you are populating the value list.

```
valueList.ValueListItems.Add 1, "One"  
valueList.ValueListItems.Add 2, "Two"  
valueList.ValueListItems.Add 3, "Three"
```

3. Make sure to set the display style of the ValueList so that it shows the **DisplayText**, and saves the **DataValue**. Also note that you need to set the column's **Type** to **DropDownList** and for convenience the **DisplayLayout's CellClickActionDefault** can be set to **Edit**.

```
valueList.DisplayStyle = Infragistics.WebUI.UltraWebGrid.ValueListDisplayStyle.  
DisplayText  
UltraWebGrid1.DisplayLayout.Bands(0).Columns.FromKey("Notes").Type = Infragistics.  
WebUI.UltraWebGrid.ColumnType.DropDownList  
UltraWebGrid1.DisplayLayout.CellClickActionDefault = Infragistics.WebUI.UltraWebGrid.  
CellClickAction.Edit
```

4. Now that the ValueList is configured, associate the ValueList with a column

```
UltraWebGrid1.DisplayLayout.Bands(0).Columns.FromKey("Notes").ValueList = valueList
```

Using Styles to Change Grid Appearance

Select one of the following sections:

- [Available Style Objects](#)
- [Style-Related Properties](#)
- [Style-Related Methods](#)

The appearance of the UltraWebGrid can be manipulated in many different ways. Style objects are provided off of several of the grid's objects so you can create a grid with the exact look and feel you desire.

UltraWebGrid is loaded with a number of different Style objects, each with numerous properties. The following is a list of Style objects included within the UltraWebGrid and a synopsis of their use:

Available Style Objects

- **GridItemStyle** - The GridItemStyle class handles properties and methods directly related to the appearance of an object inside the grid. GridItemStyles are applied to the cells, footers and column headers of the grid.
- **UltraGridStyle** - Class for implementing CSS style attributes and functionality.
- **CellStyle** - Contains the style properties for the cells of the column.
- **HeaderStyle** - Contains the style properties for the header section of the column.
- **RowSelectorStyle** - Contains the style properties for the row selector.
- **RowAlternateStyle** - Contains the style properties for alternate (even-numbered) rows in the band.
- **RowStyle** - Contains the style properties for the rows in the band.
- **SelectedRowStyle** - Contains the style properties for the rows in the band that are selected.
- **SelectedCellStyle** - Contains the style properties for the selected cell.
- **SelectedHeaderStyle** - Contains the style properties for the selected headers in the band.
- **SelectedGroupByRowStyle** - Contains the style properties for the GroupBy rows in the band that have been selected.
- **BandLabelStyle** - Returns a reference to or sets a GridItemStyle object that determines how the GroupByBox band labels are rendered on the client.
- **CellButtonStyle** - Contains the style properties for the cell buttons of the column.
- **FooterStyle** - Contains the style properties for the footer section of the column.
- **ButtonStyle** - Returns a reference to or sets a GridItemStyle object that determines how the AddNewBox button is rendered on the client.
- **GroupByRowStyle** - Contains the style properties for the GroupBy rows in the band.
- **EditCellStyle** - Specifies the style applied to the cell when it is in edit mode.

There are also default Styles that can be set to apply to all bands, columns, rows, etc. within the grid. These defaults will be overridden by the Style applied to specific bands. For example, the RowAlternateStyleDefault will be applied to all even-numbered rows throughout the grid unless a RowAlternateStyle is applied to a specific grid object. Then the properties set for the particular RowAlternateStyle will be used for that band while the other bands will still inherit from the RowAlternateStyleDefault object.

The most important aspect of the Styles objects are the properties you can set off of them. Here is a list of properties contained within the Style objects:

Style-Related Properties

- **BackColor** - Returns or sets a Color object that specifies the background color.
- **BackgroundImage** - Returns or sets the filename of an image to be displayed in the background of an object.
- **BorderColor** - Gets or sets the color of the border.
- **BorderStyle** - Gets or sets the style of the border.
- **BorderWidth** - Gets or sets the width of the border.
- **Container** - Gets the System.ComponentModel.IContainer that contains the System.ComponentModel.Component.
- **CSSClass** - Gets or sets the CSS class rendered by the Web server control on the client.
- **Cursor** - Returns or sets the type of cursor displayed for an object when the mouse pointer is over it.
- **CustomRules** - Returns or sets a string of text that determines additional CSS rules to apply to the object.
- **Font** - Returns a reference to or sets the Font object that specifies the font of the text.
- **ForeColor** - Returns or sets a Color object that specifies the foreground color.
- **HasMargin** - Read-only property that returns a boolean reflecting whether the Margin property has been set.
- **HasPadding** - Read-only property that returns a boolean reflecting whether the Padding property has been set.
- **Height** - Gets or sets the height.
- **HorizontalAlign** - Returns or sets a value that specifies the horizontal alignment of the text associated with a GridItemStyle object.
- **Margin** - Returns a reference to the Margin object that determines the amount of space for an object's margins.
- **Padding** - Returns a reference to the Padding object that determines the amount of space to insert between an object and its margin.
- **Site** - Gets information about the Web site to which the server control belongs.
- **VerticalAlign** - Returns or sets a value that specifies the vertical alignment of the text associated with a GridItemStyle object.
- **Width** - Gets or sets the width.
- **Wrap** - Returns or sets a Boolean value that determines whether the text associated with a GridItemStyle object is wrapped.

Style-Related Methods

- **ApplyStyle** - Copies any nonblank elements of the specified style to the Web control, overwriting any existing style elements of the control. This method is primarily used by control developers.
- **MergeWith** - Combines the style properties of the specified System.Web.UI.WebControls.Style with the instance of the System.Web.UI.WebControls.Style class that this method is called from.

- **CreateControlStyle** - Creates the style object that is used internally by the System.Web.UI.WebControls.WebControl class to implement all style related properties. This method is used primarily by control developers.

Also, the **HasStyle** property (and variations of it) can be used to determine if the object already has a Style applied.

For additional information, consult the topic entitled [Create and Apply Styles](#).

Create and Apply Styles

This topic assumes you have a bound grid with at least one row of data.

1. To create a **Style** object, you call the default constructor of the **Style** class. Once the Style exists, you can set its properties to the values that you want.

```
Dim style as GridItemStyle
style = new GridItemStyle()
```

2. Once you have created a Style object, you can access its properties, such as **BackColor** and **ForeColor**.

```
style.BackColor = System.Drawing.Color.Red
style.ForeColor = System.Drawing.Color.White
style.BorderStyle = BorderStyle.Groove
style.BorderWidth = New Unit("3px")
style.BackgroundImage = "images/landscape.jpg"
style.Font.Bold = true
```

3. You can then apply this Style to almost any object in the grid. For example, if you always want the Pager area to appear with white text on a red background, you can apply the Style you just created to the Pager object of the grid. You can also apply this style to other objects as well.

```
UltraWebGrid1.DisplayLayout.Pager.Style = style
UltraWebGrid1.DisplayLayout.AddNewBox.Style = style
UltraWebGrid1.DisplayLayout.GroupByBox.Style = style
```

4. You can apply the same settings to the RowSelectors by setting the RowSelectorStyleDefault to the same Style object.

```
UltraWebGrid1.DisplayLayout.RowSelectorStyleDefault = style
```

5. You could have achieved the same effect by altering the RowSelectorStyleDefault and the Pager directly.

```
UltraWebGrid1.DisplayLayout.Pager.Style.BackColor = System.Drawing.Color.Red
UltraWebGrid1.DisplayLayout.Pager.Style.ForeColor = System.Drawing.Color.White
UltraWebGrid1.DisplayLayout.RowSelectorStyleDefault.BackColor = System.Drawing.Color.Red
UltraWebGrid1.DisplayLayout.RowSelectorStyleDefault.ForeColor = System.Drawing.Color.White
```

6. However, there is a big advantage to the first method of creating a Style object and applying it. If you use the first method, you can change the properties of the Style object and the changes will carry over to all the objects the Style is applied to.

```
style.BackColor = System.Drawing.Color.Blue
```

After this line of code is executed, all the RowSelectors and the Pager and the GroupByBox in the grid change from red to blue.

Assign CSS Class to Style

Each of the style properties permits the use of Cascading StyleSheets to specify formatting.

Using code similar to:

```
grid.Bands(0).Columns(0).CellStyle.CssClass = "Headline"
```

any object in the UltraWebGrid hierarchy that has a style property, also has a style.CssClass property. Setting this value to a CSS class name will use the stylesheet's formatting definitions, instead of the default formatting definitions.

Set Border Style on the Active Row

Using the **ActivationObject** property of the **DisplayLayout** object you can choose to highlight the active row with a different border style.

In Visual Basic:

```
Me.UltraWebGrid1.DisplayLayout.ActivationObject.BorderColor = Color.Red  
Me.UltraWebGrid1.DisplayLayout.ActivationObject.BorderStyle = BorderStyle.Outset  
Me.UltraWebGrid1.DisplayLayout.ActivationObject.BorderWidth = Unit.Pixel(3)
```

In C#:

```
this.UltraWebGrid1.DisplayLayout.ActivationObject.BorderColor = Color.Red;  
this.UltraWebGrid1.DisplayLayout.ActivationObject.BorderStyle = BorderStyle.Outset;  
this.UltraWebGrid1.DisplayLayout.ActivationObject.BorderWidth = Unit.Pixel(3);
```

Set Alternate Row Colors

Use the **BackColor** property of the RowAlternateStyle object to set alternating row colors.

In Visual Basic:

```
Me.UltraWebGrid1.DisplayLayout.RowAlternateStyleDefault.BackColor = Color.Red
```

In C#:

```
this.UltraWebGrid1.DisplayLayout.RowAlternateStyleDefault.BackColor = Color.Red;
```


Change The Height Of A Single Row

To set the height of an individual Row, use the **Row.Style.Height** property. This property uses web units to determine the height of the row.

You can allow your users to change the height of a single row without affecting the rest of the rows in the band by using the **RowSizingDefault** property of the DisplayLayout object or the **RowSizing** property of the Band object. Setting the `RowSizingDefault` property on a Band overrides the default setting on the **DisplayLayout** and only applies to rows within the Band.

In Visual Basic (inside the InitializeLayout event handler):

```
e.Layout.RowSizingDefault = Infragistics.WebUI.UltraWebGrid.RowSizing.Free
```

In C# (inside the InitializeLayout event handler):

```
e.Layout.RowSizingDefault = Infragistics.WebUI.UltraWebGrid.RowSizing.Free;
```

Hide Row Selectors

The following code demonstrates how to hide the row selectors in a grid.

In Visual Basic:

```
Me.UltraWebGrid1.DisplayLayout.RowSelectorsDefault = Infragistics.WebUI.UltraWebGrid.  
RowSelectors.No
```

In C#:

```
this.UltraWebGrid1.DisplayLayout.RowSelectorsDefault = Infragistics.WebUI.UltraWebGrid.  
RowSelectors.No;
```

Making Column Headers and Footers Stationary

Column Header and Footer areas can be made stationary on the client while the row data scrolls vertically. To turn on this feature use the `DisplayLayout.StationaryMargins` property.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.StationaryMargins = Infragistics.WebUI.UltraWebGrid.  
StationaryMargins.HeaderAndFooter
```

In C#:

```
UltraWebGrid1.DisplayLayout.StationaryMargins = Infragistics.WebUI.UltraWebGrid.  
StationaryMargins.HeaderAndFooter;
```

Display a Picture in a Grid Cell

This topic assumes you have a bound grid with at least one row of data.

You must set the **Background Image** property of the cell's Style object to a URL of an image. You also should set the **CustomRules** property of the Style object as well so that the image is not repeated within the cell area of the browser.

1. Place the above snippets into the **InitializeRow** event of the UltraWebGrid.

```
e.Row.Cells.FromKey("ImageCol").Style.CustomRules = "Background-repeat:no-repeat"  
e.Row.Cells.FromKey("ImageCol").Style.BackgroundImage = "images/landscape.jpg"
```

Change Cell Appearance Based On Value

Two scenarios which you may want to change the color of a cell could be when the cell is displayed initially, as well as after the user changes the contents of the cell.

1. If you want the BackColor of the cell to be changed only after a user modifies the cell's contents on the client, you can use the **AfterCellUpdateHandler** of the Client-SideEvents object of UltraWebGrid. Create a JavaScript function as follows and set the function name to "AfterCellUpdate" in the **ClientSideEvents** property at design time.

In JavaScript:

```
function AfterCellUpdate(tableName, itemName)
{
    var cell = igtbl_getElementById(itemName);
    if(cell.innerHTML == "Test")
    {
        cell.runtimeStyle.backgroundColor = "Red";
        cell.style.backgroundColor = "Red";
    }
    else {
        cell.runtimeStyle.backgroundColor = "Green";
        cell.style.backgroundColor = "Green"; return 0;
    }
}
```

Be sure to declare the JavaScript function with the two parameters indicated above.

The changed color values will not be updated to the server side, so they will not persist if the page is submitted back to the server and re-rendered.

2. If you want the bgcolor of the cell to be set depending on it's value when the grid initially loads, you can use the **InitializeRow** event of the UltraGrid.

In Visual Basic:

```
If e.Row.Cells.FromKey("Subject").Value = "Test" Then
    e.Row.Cells.FromKey("Subject").Style.BackColor = System.Drawing.Color.Red
End If
```

Set Cell Borders and Grid Lines

To configure the gridlines or cell borders for UltraWebGrid, 2 steps are used. First the DisplayLayout.GridLinesDefault is set to GridLines.Horizontal, GridLines.Vertical or GridLines.Both. Then the style of the gridlines is chosen by using the RowStyleDefault property of the DisplayLayout object. Here the border width, BorderColor and BorderStyle properties can be set to customize the appearance of the border lines.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.GridLinesDefault = UltraGridLines.Both
UltraWebGrid1.DisplayLayout.RowStyleDefault.BorderColor = Color.Gray
UltraWebGrid1.DisplayLayout.RowStyleDefault.BorderStyle = BorderStyle.Groove
UltraWebGrid1.DisplayLayout.RowStyleDefault.BorderWidth = New Unit(3)
```

In C#:

```
UltraWebGrid1.DisplayLayout.GridLinesDefault = UltraGridLines.Both;
UltraWebGrid1.DisplayLayout.RowStyleDefault.BorderColor = Color.Gray;
UltraWebGrid1.DisplayLayout.RowStyleDefault.BorderStyle = BorderStyle.Groove;
UltraWebGrid1.DisplayLayout.RowStyleDefault.BorderWidth = new Unit(3);
```

Using the Activation Object

The Activation object is used to specify the attributes of the object that is currently active. The Active object is the object that currently has the input focus. In UltraWebGrid, only Row and Cell objects become active. When a cell becomes active, the row that contains it becomes the active row.

There can only be one active row and one active cell at a time. Activation is not the same thing as selection. While multiple rows or cells can be selected, only one row and one cell are the active ones. Typically, the active row and cell serve as the anchor point for a selection.

Use the Activation object to distinguish the active cell and row from other cells and rows in the grid that are only selected, or neither active nor selected. The attributes of the Activation object are applied to the ActiveCell and the ActiveRow objects when a cell and a row become active. Properties of the Activation object include **BorderColor**, **BorderStyle** and **BorderWidth**.

You can also use the **AllowActivation** property of the object to toggle the user's ability to activate rows and cells. With activation disabled, the user cannot edit cell contents or add new rows.

Remove Scrollbars and Set Max Width and Height

In order to get rid of the scrollbars completely and set the maximum width and height for UltraWebGrid use the following code:

In Visual Basic:

```
Me.UltraWebGrid1.Width=Unit.Percentage(100)
Me.UltraWebGrid1.Height=Unit.Percentage(100)
Me.UltraWebGrid1.DisplayLayout.FrameStyle.CustomRules = "table-layout:auto"
```

In C#:

```
this.UltraWebGrid1.Width=Unit.Percentage(100);
this.UltraWebGrid1.Height=Unit.Percentage(100);
UltraWebGrid1.DisplayLayout.FrameStyle.CustomRules = "table-layout:auto";
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #2**.

Row Selection and The Active Row

The UltraWebGrid uses the concept of an "active row" when referencing the row that is current in the grid. The active row is determined by one of several factors. Any row that has input focus within one of its cells and is in edit mode is automatically the active row. The active row can also be determined by selection. If the user clicks on a row to select it, that row becomes the active row. If the user is performing a range selection, the row that is defining the farthest extent of the range is the active row. (The range is determined by the relationship of the position of the active row to that of the "anchor" row, or the row that was active when the range selection was initiated.)

There can be several selected rows within a grid, but, at any given moment, there can be only one active row.

The active row can be set by user action or through code. The following code will set the active row of the UltraWebGrid:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ActiveRow = UltraWebGrid1.Rows(0)
```

In C#:

```
UltraWebGrid1.DisplayLayout.ActiveRow = UltraWebGrid1.Rows[0];
```

The appearance of the active row can be manipulated as well. The ActiveRow object has its own Style object assigned to it. Changing properties on this object will only affect the active row. If you would like to set properties for the appearance of all rows, go through the RowStyle object. The following code will set the BackColor property of the ActiveRow:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ActiveRow.Style.BackColor = Color.Red
```

In C#:

```
UltraWebGrid1.DisplayLayout.ActiveRow.Style.BackColor = Color.Red;
```

You can set rows as selected through code as well. Each row has a Selected property. Setting this property to true will add this row to the SelectedRows collection of the UltraWebGrid. The SelectedRows collection stores all cells whose Selected property is true. The following code will select the row and then clear the SelectedRows collection:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.Rows(0).Selected = True  
UltraWebGrid1.DisplayLayout.SelectedRows.Clear()
```

In C#:

```
UltraWebGrid1.DisplayLayout.Rows[0].Selected = true;  
UltraWebGrid1.DisplayLayout.SelectedRows.Clear();
```

The appearance of the selected rows can be changed through the SelectedRowStyleDefault object. Setting properties through this object will affect all rows that are currently selected. The following code will change the BackColor on only the rows within the UltraWebGrid that are selected:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.SelectedRowStyleDefault.BackColor = Color.Red
```

In C#:

```
UltraWebGrid1.DisplayLayout.SelectedRowStyleDefault.BackColor = Color.Red;
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #4**.

How To Change The Active Row

The following code demonstrates how to change the active row.

In Visual Basic:

```
Dim objRow As Infragistics.WebUI.UltraWebGrid.UltraGridRow
objRow = Me.UltraWebGrid1.Rows(5)
Me.UltraWebGrid1.ActiveRow = objRow
```

In C#:

```
Infragistics.WebUI.UltraWebGrid.UltraGridRow objRow;
objRow = this.UltraGrid1.Rows[5];
this.UltraWebGrid1.ActiveRow = objRow;
```

Expand and Collapse Rows on Server-Side

There are several different ways one may expand/collapse a row in UltraWebGrid. The following code demonstrates several options for expanding rows.

In Visual Basic:

```
'expands the current row and its children as well
Me.UltraWebGrid1.Rows(0).Expand(True)
'expands the current row
Me.UltraWebGrid1.Rows(0).Expand(False)
'expands all the ancestors of the current row
Me.UltraWebGrid1.Rows(0).ExpandAncestors()
'expands only the current row
Me.UltraWebGrid1.Rows(0).Expanded = True

'collapses the current row and its children
Me.UltraWebGrid1.Rows(0).Collapse(True)
'collapses only the current row
Me.UltraWebGrid1.Rows(0).Collapse(False)
```

In C#:

```
//expands the current row and its children as well
this.UltraWebGrid1.Rows[0].Expand(true);
//expands the current row
this.UltraWebGrid1.Rows[0].Expand(False);
//expands all the ancestors of the current row
this.UltraWebGrid1.Rows[0].ExpandAncestors();
//expands only the current row
this.UltraWebGrid1.Rows[0].Expanded = true;

//collapses the current row and its children
this.UltraWebGrid1.Rows[0].Collapse(true);
//collapses only the current row
this.UltraWebGrid1.Rows[0].Collapse(false);
```

Scrolling The Grid on Client-Side

This article describes the use of the `scrollToView` function.

To scroll an object into view use the following JavaScript function:

function igtbl_scrollToView(gn, child) - scrolls a child object of the grid (row or cell) into view.

The *gn* parameter is the name of the grid. The *child* parameter is the DOM object you want to place into view. The *child* parameter has to be a child of the grid.

For example if you want to bring recently activated row into view you could use this code:

```
function AfterRowActivate(gn, rowID)
{
    var row=igtbl_getElementById(rowID);
    igtbl_scrollToView(gn, row);
}
```

Moving Columns Programmatically

If you want the columns of data in your grid to be in a different order from the order in which it resides in the database, you must manually move the columns.

Open a project that has a grid on it. The "Bind to a DataTable" tutorial is a good place start if you don't have any other projects. Add an ASP button in the design view, and double click it to see the code-behind.

In the button's **Click** event add this code:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.Bands(0).Columns(2).Move(0)
```

In C#:

```
UltraWebGrid1.DisplayLayout.Bands[0].Columns[2].Move(0);
```

Run the project. Click the button, and a third column will be move to the first column, and the other columns will shift to the right.

Since moving a column bumps all the columns over one, if you are moving multiple columns it is a good idea to start with whichever one you want to appear leftmost. For example:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.Bands(0).Columns.FromKey("SomeColumn").Move(0)
UltraWebGrid1.DisplayLayout.Bands(0).Columns.FromKey("NextColumn").Move(1)
UltraWebGrid1.DisplayLayout.Bands(0).Columns.FromKey("AnotherColumn").Move(2)
```

In C#:

```
UltraWebGrid1.DisplayLayout.Bands[0].Columns.FromKey("SomeColumn").Move(0);
UltraWebGrid1.DisplayLayout.Bands[0].Columns.FromKey("NextColumn").Move(1);
UltraWebGrid1.DisplayLayout.Bands[0].Columns.FromKey("AnotherColumn").Move(2);
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #2**.

Cell Selection and The Active Cell

The UltraWebGrid uses the concept of an "active cell" when referencing the cell that is current in the grid. The active cell is determined by one of several factors. Any cell that has input focus and is in edit mode is automatically the active cell. The active cell can also be determined by selection. If the user clicks on a cell to select it, that cell becomes the active cell. If the user is performing a range selection, the cell that is defining the farthest extent of the range is the active cell. (The range is determined by the relationship of the position of the active cell to that of the "anchor" cell, or the cell that was active when the range selection was initiated.)

There can be several selected cells within a grid, but, at any given moment, there can be only one active cell.

The active cell can be set by user action or through code. The following code will set the active cell of the UltraWebGrid:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ActiveCell = UltraWebGrid1.Rows(0).Cells(0)
```

In C#:

```
UltraWebGrid1.DisplayLayout.ActiveCell = UltraWebGrid1.Rows[0].Cells[0];
```

The appearance of the active cell can be manipulated as well. The ActiveCell object has its own Style object assigned to it. Changing properties on this object will only affect the active cell. If you would like to set properties for the appearance of all cells, go through the RowStyle object. The following code will set the BackColor property of the ActiveCell:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ActiveCell.Style.BackColor = Color.Red
```

In C#:

```
UltraWebGrid1.DisplayLayout.ActiveCell.Style.BackColor = Color.Red;
```

You can set cells as selected through code as well. Each cell has a Selected property. Setting this property to true will add this cell to the SelectedCells collection of the UltraWebGrid. The SelectedCells collection stores all cells whose Selected property is true. The following code will select the cell and then clear the SelectedCells collection:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.Rows(0).Cells(0).Selected = True  
UltraWebGrid1.DisplayLayout.SelectedCells.Clear()
```

In C#:

```
UltraWebGrid1.DisplayLayout.Rows[0].Cells[0].Selected = true;  
UltraWebGrid1.DisplayLayout.SelectedCells.Clear();
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #3**.

Protect Cell From Editing on Client-Side

Sometimes you only want certain cells in a column/row editable. However, setting the **AllowUpdate** property of the column will affect every cell in the column and there isn't an **AllowUpdate** property for a row.

Here is a method for implementing the functionality of editing only certain cells when the column's **AllowUpdate** property is set to false. If you cancel the **BeforeEnterEditMode** event on the client by returning true, the user will not be able to edit that cell.

First, add a handler for the client-side event **BeforeEnterEditMode**. This property can be set either in the Property Pages under displaylayout -> ClientsideEvents -> BeforeEnterEditMode or in code:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ClientsideEvents.BeforeEnterEditModeHandler = "BeforeEdit"
```

In C#:

```
UltraWebGrid1.DisplayLayout.ClientsideEvents.BeforeEnterEditModeHandler = "BeforeEdit";
```

This event receives two parameters, *gridname* and *cellid*. You can return true to cancel the event which will stop the cell from being edited. The following JavaScript demonstrates how to cancel the event for any row whose cell in column one has a value of true.

```
function BeforeEdit(gridname, cellid)
{
    //get the cell that is about to be edited
    var row=igtbl_getRowById(cellid);
    //get the value for column 1 in the current row, gets the value by
    //getting the cell's row object then goes into the row's cell array
    var value=row.getCell(1).getValue();
    //check if the value is true, if so then don't allow editing of the column,
    //cancel event by returning true, else do nothing and allow editing
    if(value==true)
    {
        return true;
    }
    //or in this case since editing is based on the boolean value
    //you could just return the value and omit the if statement.
}
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #3**.

Begin Editing Cell on Cell Click

To have the grid go into edit mode for a cell when it is clicked, use the **CellClickActionDefault** property of the **DisplayLayout** object. This assumes that the **AllowUpdateDefault** property of **DisplayLayout** has also been set to **Yes**

In Visual Basic:

```
Me.UltraWebGrid1.DisplayLayout.AllowUpdateDefault = AllowUpdate.Yes  
Me.UltraWebGrid1.DisplayLayout.CellClickActionDefault = CellClickAction.Edit
```

In C#:

```
this.UltraWebGrid1.DisplayLayout.AllowUpdateDefault = AllowUpdate.Yes;  
this.UltraWebGrid1.DisplayLayout.CellClickActionDefault = CellClickAction.Edit;
```

Embedding WebTextEditors in WebGrid

All of the **WebDataInput** editors can be embedded into the Infragistics **WebGrid** as embedded editors.

To add any of the **WebDataInput** editors as editors in **WebGrid** cells:

1. Add the **WebDataInput** editors to the ASPX page that you are going to use in the cells of a **WebGrid**.
2. Set the **BorderStyle** of the WebDataInput editors to **None**.
3. Add a **WebGrid** to an ASPX page.
4. Set the **DataSource** and **DataMember** properties to your data source if you are using the WebGrid in bound mode, or add columns manually if you are using WebGrid in unbound mode.
5. Change the **AllowUpdateDefault** property of the WebGrid to **Yes**.
6. Right click on the WebGrid, and select **Edit Columns** from the contextual menu.
7. From the **Members** pane on the right of the **Columns Collection Editor**, select the field you wish to use the editor with.
8. From the **EditorControlID** property, select the **ID** of the editor on the ASPX page you wish to use for the selected field.
9. Change the **Type** property to **Custom**.

Columns Collection Editor

Members:

0	Mask
1	Double
2	Range: -9..90000
3	Short
4	Date
5	Range: 1/1/1920-12/31/...

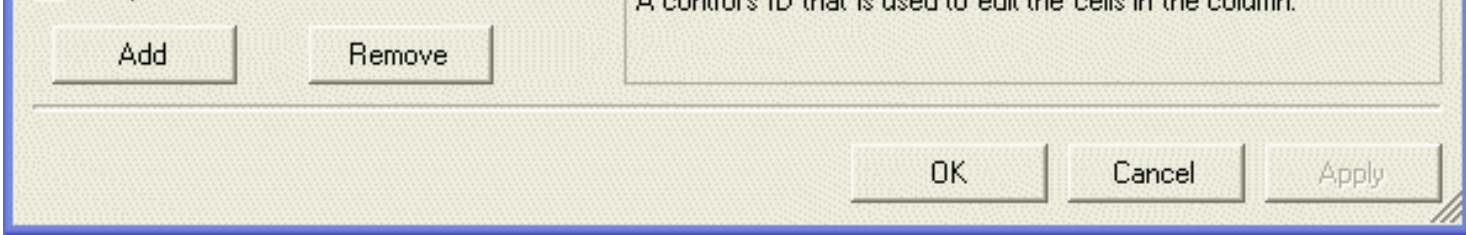
Properties:

AllowUpdate	NotSet
Case	Unchanged
CellButtonDisplay	OnMouseEnter
CellMultiline	NotSet
DefaultValue	
EditorControlID	WebMaskEdit1
Format	WebDateTimeEdit2
HeaderClickAction	WebNumericEdit1
Hidden	WebCurrencyEdit1
HTMLEncodeContent	WebMaskEdit1
MergeCells	WebNumericEdit2
ServerOnly	WebDateTimeEdit1
Type	Custom
Validators	(Collection)
Data	
BaseColumnName	
DataType	System.String
FieldLen	0
IsBound	False
Key	
NullText	
ValueList	
Headers and Footers	
FooterStyle	

Templated column

Add Remove

EditorControlID
A control's ID that is used to edit the cells in the column.



Repeat steps 7 thru 9 for each editor you want to use in the cells of the WebGrid.

Add a Custom Editor Control to the WebGrid

You can use a custom editor control as an editor in the WebGrid. In order to serve as a custom editor control, the control you use must implement the `IProvidesEmbeddableEditor` interface. Take the following steps to set this up:

1. Add a WebGrid to a web form (if it does not already exist) and set up any data binding and column structure for the grid.
2. Allow users to edit the grid data by setting the **AllowUpdateDefault** property to true for the control. You can optionally leave the control as read-only and set just the column(s) that will contain the custom control(s) to be editable.
3. Place the control that will serve as the custom editor onto the web form at design-time, or create it through code in the code-behind for the web form. You must use a control that is capable of being used as a custom editor (implements the `IProvidesEmbeddableEditor` interface.)
4. Change the **Type** property of the column that will contain the custom editor to "Custom".
5. Set the **EditorControlID** property of the column to the ID of the custom editor control. If you have placed the custom editor control on the web form at design-time, you can select the name of the control from the dropdown that appears when editing the **EditorControlID** property in the Visual Studio property sheet. This dropdown shows all controls on the web form capable of being used as custom editors.
6. You can optionally set the **Format** property of the column so that the data displayed in the column will match the data shown by the editor. For example, if the custom editor you're using is the Infragistics DateChooser, set the column's **Format** to "MM/dd/yyyy" to have the date formatted in the column to match the editor's display.
7. Run the application and try out the editor by placing a cell in the column into edit mode. (Double-click the cell by default).

Multi-Line Cell Editing

By default, the browser will adjust the height of a row to whatever size is required to view all text of the largest cell of the row. If the height of a row is explicitly set or if the `DisplayLayout.RowHeightDefault` property is set to a specific value, then all text that exceeds the height will be cut off.

In order to edit cells that have large amounts of text, the `Column.CellMultiline` property can be set to `CellMultiLine.Yes`. With this property turned on, editing for the cell takes place within a `TextArea` control that can be scrolled to view and edit multiple lines of text.

In Visual Basic:

```
UltraWebGrid1.Columns.FromKey("Description").CellMultiLine = CellMultiLine.Yes
```

In C#:

```
UltraWebGrid1.Columns.FromKey("Description").CellMultiLine = CellMultiLine.Yes;
```

Change ValueLists on a Per Row Basis

While supplying different valuelists on different rows is not an feature intrinsic to the grid, it is easy to simulate this functionality. We can change the makeup of a valuelist on the client-side programmatically. Suppose there is a grid with three columns. In the first and second column, we can choose teams, say baseball teams who are playing each other. In the third column, we will choose the team we think will win. Naturally, the valuelist in the third column should show only two entries, those teams shown in the first two columns.

To obtain the Visual Studio.NET project files associated with this topic, [click this link](#). (This link points to a resource on the Infragistics website and requires an active Internet connection.)

WebForm1.aspx.vb

In the form load event you have the following code:

In Visual Basic:

```
' Add three columns
UltraWebGrid1.Columns.Add("Team 1", "Team 1")
UltraWebGrid1.Columns.Add("Team 2", "Team 2")
UltraWebGrid1.Columns.Add("Winner", "Winner")

' Create a ValueList and add some teams
Dim vlist As New ValueList()
vlist.ValueListItems.Add("Angels")
vlist.ValueListItems.Add("Athletics")
vlist.ValueListItems.Add("Marlins")
vlist.ValueListItems.Add("Orioles")
vlist.ValueListItems.Add("Phillies")
vlist.ValueListItems.Add("Pirates")
vlist.ValueListItems.Add("Rangers")
vlist.ValueListItems.Add("Red Sox")
vlist.ValueListItems.Add("White Sox")

' Set the ValueList for each of the three columns
UltraWebGrid1.Columns(0).ValueList = vlist
UltraWebGrid1.Columns(1).ValueList = vlist
UltraWebGrid1.Columns(2).ValueList = vlist

' The three columns must also be of type "DropDownList"
UltraWebGrid1.Columns(0).Type = ColumnType.DropDownList
UltraWebGrid1.Columns(1).Type = ColumnType.DropDownList
UltraWebGrid1.Columns(2).Type = ColumnType.DropDownList

' Add 6 rows just to give you room to play around
UltraWebGrid1.Rows.Add()
UltraWebGrid1.Rows.Add()
UltraWebGrid1.Rows.Add()
UltraWebGrid1.Rows.Add()
UltraWebGrid1.Rows.Add()
UltraWebGrid1.Rows.Add()

' Make sure we can update the grid
UltraWebGrid1.DisplayLayout.AllowUpdateDefault = AllowUpdate.Yes
```

With the comments, most of this is self explanatory.

WebForm1.aspx

In the .aspx page, click the HTML view button in the lower left corner of the Designer. Notice in the <HEAD> there is the following JavaScript block:

```
function BeforeEnterEditMode(tableName, cellName)
{
    var col = igtbl_getColumnById(cellName);
    if (col.Key != "Winner")
        return;

    var row = igtbl_getRowById(cellName);

    var vlist = new Array(2);
    vlist[0] = new Array(row.getCell(0).getValue(), row.getCell(0).getValue());
    vlist[1] = new Array(row.getCell(1).getValue(), row.getCell(1).getValue());

    col.ValueList = vlist
}
```

This is where the work gets done. This may look strange if you have never worked with JavaScript or our client-side events before. Here is what is happening. This code is handling the **BeforeEnterEditMode** event. This event will get called after the user double-clicks a cell in the grid to edit it, but before it actually goes into edit mode.

1. First, the Column that contains the cell is retrieved. If this cell is not in the "Winner" column, no action is needed, so the code returns from this function. ("return" is equivalent to VB's "Exit Sub.") Then the code gets the row of the current cell.
2. Since a valuelist is represented simply as an array of arrays, you can construct your valuelist as shown here. Once the valuelist has been constructed, you assign it to the Column's valuelist.
3. The items in the valuelist are represented as an array because they have both a value and a display text. For the purpose of simplifying this sample, the name of the baseball team is used as both the value and the display text.
4. The last step is to tell the grid that this javascript function exists, so that the function gets called at the appropriate time. To do this, you need to switch back to the Design view, click on the grid, and navigate in the property sheet to DisplayLayout -> ClientSideEvents -> BeforeEnterEditModeHandler. Set this value to "BeforeEnterEditMode". Note that if you had called your function "MakeTheWinnerValueListOnlyShowTwoTeams", you would set BeforeEnterEditModeHandler to that value. That is, you can call the function anything you want, as long as you tell the grid what that name is.
5. If you started this project from scratch, you may also want to set some colors on the grid so that you can see where the cells are. To do this quickly, right click on the grid and choose Auto-Format. Choose a format here, for example, "Contrast II".
6. Run the project. Double-click in the first cell, choose a team. Double-click in the second cell, choose a team. Double click in the third cell and notice that the only two teams to choose from are the two in the first two columns. Continue to play around in the rest of the rows.

Loop Through The Rows of a Band

To loop through all rows of a band other than band 0, first loop through all rows of the parent Band and then through the children of each row in the parent Band.

To loop through all rows in Band 1 use the following code.

In Visual Basic:

```
Dim en As UltraGridRowsEnumerator
en = UltraGrid1.Bands[1].GetRowEnumerator()
While en.MoveNext() != Nothing
    Dim row As UltraGridRow
    row = (UltraGridRow)en.Current
End While
```

In C#:

```
UltraGridRowsEnumerator en = UltraGrid1.Bands[0].GetRowsEnumerator();
while(en.MoveNext())
{
    UltraGridRow row = (UltraGridRow)en.Current;
}
```


Add Rows Without Default Values

When you add a row on the server side using the **Band.Addnew()** method, default values for the row are provided. If you would like to add a row without any default values, you must use a different method.

Open a project (or a copy of a project) that has a form containing a WebGrid. (The "Bind to a DataTable" tutorial is a good place start, if you don't have any other projects.) Add an ASP button in the Design view, and double-click it to see the code.

In the button's **Click** event add this code:

In Visual Basic:

```
UltraWebGrid1.Rows.Add()
```

In C#:

```
UltraWebGrid1.Rows.Add();
```

Run the project. Click the button, and a new row will be added to the grid.

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #2**.

Add Rows To Database Without AutoNumber

This article discusses how to add rows to your database from the UltraWebGrid where this is no AutoNumber field in the dataset.

If you want to add a row to your database without having an autonumber field then the **Update** event of the grid should be used. Within this event, check each row's **DataChanged** property to see if it is an added Row. The reason for this is that in the **addRowBatch** event you do not have access to values in the added row of the grid, but you do in the update event of the grid. The following code gets all of the changed rows in the grid, checks to see if it is an added row. If it is an added row, it loops through the values to create a new row in the dataset and adds them to the dataset.

In Visual Basic:

```
Private Sub UltraWebGrid1_UpdateGrid(ByVal sender As Object, ByVal e As Infragistics.WebUI.
UltraWebGrid.UpdateEventArgs) Handles UltraWebGrid1.UpdateGrid
    Dim table As DataTable
    Dim row As UltraGridRow
    'get a reference to the datatable
    table = DataSet11.Tables(e.Grid.Bands(0).BaseTableName)
    'create and get the rows enumeration for the changed rows
    Dim updatedRows As UltraGridRowsEnumerator
    updatedRows = UltraWebGrid1.Bands(0).GetBatchUpdates()
    'for each row in the Updated rows check if the current row is an addedrow, if so create
the row and add it to the dataset
    While updatedRows.MoveNext
        row = updatedRows.Current
        If row.DataChanged = DataChanged.Added Then
            Dim i As Integer
            Dim addedRow As DataRow
            addedRow = table.NewRow()
            For i = 0 To row.Cells.Count - 1
                addedRow(i) = row.Cells(i).Value
            Next
            table.Rows.Add(addedRow)
        End If
    End While
End Sub
```

In C#:

```
private void UltraWebGrid1_UpdateGrid(object sender, Infragistics.WebUI.UltraWebGrid.
UpdateEventArgs e)
{
    DataTable table;
    UltraGridRow row;
    //get a reference to the datatable
    table = dataSet11.Tables[e.Grid.Bands[0].BaseTableName];
    //create and get the rows enumeration for the changed rows
    UltraGridRowsEnumerator updatedRows;
    updatedRows = UltraWebGrid1.Bands[0].GetBatchUpdates();
    //for each row in the Updated rows check if the current row is an addedrow, if so create
the row and add it to the dataset
    while(updatedRows.MoveNext())
    {
        row = (UltraGridRow)updatedRows.Current;
        if(row.DataChanged==DataChanged.Added)
        {
            DataRow addedRow;
```

```
addedRow = table.NewRow();
for(short i = 0; i < row.Cells.Count - 1;i++)
{
    addedRow[i] = row.Cells[i].Value;
    table.Rows.Add(addedRow);
}
}
}
```

Get A Value From A Specific Cell In A Specific Row

The following code demonstrates how to get a value from a specific cell in a specific row.

In Visual Basic:

```
Dim text as string = Me.UltraWebGrid1.Rows(2).Cells(2).Text()
```

In C#:

```
string text = UltraWebGrid1.Rows[10].Cells[1].Text;
```

Updating a Column Footer

On the client side it may be necessary sometimes to update a column's footer according to changes that are made in its cells. Here is how to do so.

To update a column footer you may use the following JavaScript functions.

This one takes two parameters - cell ID and the value you'd like to put in the cell's row island footer.

```
function updateCellFooter(cellId, value)
{
    var cell=igtbl_getElementById(cellId);
    if(cell.parentNode.parentNode.nextSibling)
    {
        var footer=cell.parentNode.parentNode.nextSibling.childNodes[0].childNodes[cell.
cellIndex];
        footer.innerText=value;
    }
}
```

This function may be used if you need to update some column's footer using its ID. Note that if you use the function for the band 1 and higher it will update the footer in all row islands of the band.

```
function updateColumnFooter(columnId, value)
{
    var col=igtbl_getDocumentElement(columnId);
    if(col.length)
    {
        for(var i=0;i<col.length;i++)
        {
            if(col[i].parentNode.parentNode.nextSibling.nextSibling)
            {
                var footer=col[i].parentNode.parentNode.nextSibling.nextSibling.childNodes[0].
childNodes[col[i].cellIndex];
                footer.innerText=value;
            }
        }
    }
    else
    {
        if(col.parentNode.parentNode.nextSibling.nextSibling)
        {
            var footer=col.parentNode.parentNode.nextSibling.nextSibling.childNodes[0].
childNodes[col.cellIndex];
            footer.innerText=value;
        }
    }
}
```

If you want to keep compatibility with Netscape's products use `innerHTML` instead of `innerText`.

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #2**.

Configure the Grid with InitializeLayout

The InitializeLayout event is where the UltraWebGrid should be configured after binding to a data source. This event is fired from within the data binding logic of the control at the point where the Bands and Columns of the grid have been loaded but before any rows of data have been bound.

This event is useful for adding unbound columns to the grid as well as for hiding any particular columns or bands. It can also be used to set the properties of the bands and columns in a single, centralized location.

In Visual Basic:

```
Private Sub UltraGrid1_InitializeLayout(ByVal sender As Object,
ByVal e As Infragistics.WebUI.UltraWebGrid.LayoutEventArgs) Handles
UltraWebGrid1.InitializeLayout
    With e.Layout.Bands(0).Columns
        .FromKey("ManufacturerName").HeaderText = "Manufacturers"
        .FromKey("ManufacturerID").Hidden = True
        ' Add an unbound column to Bands(0) for the car's image
        .Insert(0, "CarImage")
        .FromKey("CarImage").HeaderText = "Image"
    End With
End Sub
```

In C#:

```
private void UltraWebGrid1_InitializeLayout(font color="0000ff"
size="2">object sender, Infragistics.WebUI.UltraWebGrid.LayoutEventArgs
e)
{
    e.Layout.Bands[0].Columns.FromKey("ManufacturerName").HeaderText
= "Manufacturers";
    e.Layout.Bands[0].Columns.FromKey("ManufacturerID").Hidden =
true;
    // Add an unbound column to Bands(0) for the car's image
    e.Layout.Bands[0].Columns.Insert(0, "CarImage");
    e.Layout.Bands[0].Columns.FromKey("CarImage").HeaderText =
"Image";
}
```

Using the InitializeRow Event

The **InitializeRow** event is fired as each row is added to the rows collection. It provides an opportunity to alter the contents of cells and to populate cells in unbound columns. It can also be used to set styles on individual cells.

There are numerous topics within the Help files and the Knowledge Base that show how this event can be used.

For more examples on utilizing this event, please refer to the following topics:

- [Change Cell Appearance Based On Value](#)
- [Create Column Footers and Column Totals](#)
- [Display a Picture in a Grid Cell](#)
- [Perform data validation on cells as when loaded into the WebGrid](#)
- [How to add hyperlinks to the UltraWebGrid](#)

Access New Dropdown Selection on Client-Side

How can the newly selected item in the dropdown be accessed on the client? The ValueListSelChange handler was provided for this functionality.

First, add a handler for the client-side event, ValueListSelChangeHandler. The property for the event can be set either in the Property Pages under displaylayout -> ClientsideEvents -> ValueListSelChangeHandler or in code:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ClientsideEvents.ValueListSelChangeHandler = "ValueListSelChange"
```

In C#:

```
UltraWebGrid1.DisplayLayout.ClientsideEvents.ValueListSelChangeHandler = "ValueListSelChange";
```

This event receives three parameters: gridname, row id and the select element id. The following JavaScript illustrates how to retrieve the newly selected value.

```
//displays the newly selected item
function ValueListSelChange(gridID, ValueListID, cellID) {
    //gets the Select Element used in the combo
    var list = igtbl_getElementById(ValueListID);
    //display the current value from the Select element
    alert(list.value);
}
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #4**.

WebCombo With ValueList

The WebCombo control can be used as a dropdown selection editor inside of a cell. This capability is utilized as follows:

- With WebGrid on a form, add a WebCombo control to the form as well.
- The **BorderStyle** of the WebCombo should be set to None for an improved appearance within WebGrid cells.
- The WebCombo control can be configured and bound to data in any of the normal ways available to that control. A typical way of utilizing a cell dropdown is to display a human readable version of a column that is a foreign key of another table. In our example we will use the Products and Categories tables of the NWind.mdb database that installs with the product. For a complete working example of the code described in this article, see the ValueLists sample, second page: WebGrid with WebCombo ValueList.
- Add a database connection for the NWind database. Add OleDbDataAdapters for the Products and Categories tables. Create a dataset to contain the two tables.
- Connect the WebGrid **DataSource** property to dataSet11 and the **DataMember** to the Products table. Edit the WebGrid columns for the Products table so that the HeaderText property for the CategoryID column reads CategoryName, as this is what the effect will be once WebCombo is attached to this column.
- Connect the WebCombo **DataSource** property to dataSet11 and the **DataMember** to the Categories table. Edit the WebCombo columns collection for the Categories table so that the CategoryID column's **Hidden** property is set to True. You don't need to see this column at all since the CategoryName column is what will be used to select the value.
- Set the WebCombo **DataTextField** property to CategoryName. This tells WebCombo which column to place in the top portion when a row is selected from the dropdown.
- Set the WebCombo **DataValueField** property to CategoryID. This tells WebCombo which column to use as the actual value of the control. The **DataTextField** and the **DataValueField** are used together by WebGrid to display the first and update the second as part of database processing.
- In the code behind file, add the following code to make the connection between WebGrid and WebCombo:

In Visual Basic:

```
UltraWebGrid1.Columns.FromKey("CategoryID").AllowUpdate = AllowUpdate.Yes  
UltraWebGrid1.Columns.FromKey("CategoryID").Type = ColumnType.DropDownList  
UltraWebGrid1.Columns.FromKey("CategoryID").ValueList.WebCombo = WebCombo1
```

In C#:

```
UltraWebGrid1.Columns.FromKey("CategoryID").AllowUpdate = AllowUpdate.Yes;  
UltraWebGrid1.Columns.FromKey("CategoryID").Type = ColumnType.DropDownList;  
UltraWebGrid1.Columns.FromKey("CategoryID").ValueList.WebCombo = WebCombo2;
```

```
UltraWebGrid1.Columns.FromKey("CategoryID").AllowUpdate=AllowUpdate.Yes UltraWebGrid1.Columns.  
FromKey("CategoryID").Type=ColumnType.DropDownList UltraWebGrid1.Columns.FromKey("CategoryID").  
ValueList.WebCombo = WebCombo1
```

Handling Client-Side Events

To respond to events from within the client browser, use the `DisplayLayout.ClientSideEvents` object to enter the names of JavaScript function handlers. The function names that you specify will be called on the client when the event is raised. The parameters of the event always include the name of the grid as well as the Id of the element on which the event occurred. A few events have one or more additional parameters to convey more information.

The return value of client-side events determines whether the event is processed any further. If the return value is zero, the event processing continues normally. If the return value is non-zero, the event is, in effect, canceled, and no further processing takes place.

The JavaScript functions for client-side events can be added to the .aspx page directly.

In Visual Basic:

```
e.Layout.ClientSideEvents.MouseOverHandler = "MouseOver"
```

In C#:

```
e.Layout.ClientSideEvents.MouseOverHandler = "MouseOver";
```

JavaScript

```
function MouseOver(tableName, itemName, type)
{
    if(type == 0) { // Are we over a cell
        var cell = igtbl_getElementById(itemName);
        cell.style.cursor = 'hand';
        var label = igtbl_getElementById("Label2");
        var parts = itemName.split("_");
        if(label && parts.length>=3)
            label.innerHTML = "Current Row:" + parts[1] + " - Current Column:" + parts[2];
    }
}
```

How the Client-Side Event Model Functions

The **Events** property of the grid consists of a fixed length array of individual event arrays. Each event array consists of two values: a string that identifies a client-side JavaScript function to call when the event is raised, and a boolean value that indicates whether to raise a post-back to the server for processing of the event there. Event processing on the client will be discussed in more detail below.

Many of the events on the client that are handled by the grid can also be handled by an application using JavaScript function handlers. The name of the function handler is entered into the ClientSideEvents object on the server and rendered into the events array on the page. When the event is raised on the client, the grid checks to see if there is a handler for the event. Every handler is passed parameter information identifying which objects the event applies to. In the case of the **InitializeLayout** event, for instance, the ID of the grid is passed. In the case of an event that applies to a cell, both the ID of the grid and the ID of the cell are passed. From the passed in IDs, objects can easily be obtained using one of several utility functions. To obtain the grid object, use `igtbl_getGridById(gridId)`. To obtain a row object use `igtbl_getRowById(cellId)`.

To cancel the client-side processing of an event, all the event handler has to do is return true, or non-zero. This tells the grid logic that the application has either handled the event itself, or wishes to terminate event processing for some reason. The grid will then behave as if the event did not take place.

Along with the name of each client side event handler in the Events object is a boolean value that indicates whether or not a post-back should occur after the event completes. A post-back generally occurs if there is a handler for the event on the server page. A post-back for a server-side event can occur with or without a client-side event handler. If there is both a server-side event and client-side event defined for a particular event, then the client-side event has the option of canceling the post-back for the server-side event. It can do this by setting **CancelPostBack** equal to True on the Grid object.

If a client-side event handler wants to force a post-back to the server where there otherwise would not be one, it sets `gridObject.NeedPostBack` to True.

ClientSideEvents Class

For a list of all members of this type, see [ClientSideEvents members](#).

Inheritance Hierarchy

[System.Object](#)

[Infragistics.WebUI.Shared.WebComponentBase](#)

Infragistics.WebUI.UltraWebGrid.ClientSideEvents

Syntax

```
[Visual Basic]
Public Class ClientSideEvents
    Inherits WebComponentBase
    Implements IStateManager, ISupportRollback
```

```
[C#]
public class ClientSideEvents : WebComponentBase, IStateManager, ISupportRollback
```

```
[JScript]
public class ClientSideEvents extends WebComponentBase implements
IStateManager, ISupportRollback
```


See Also

[ClientSideEvents Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)












ClientSideEvents Class Members

[ClientSideEvents overview](#)



















Public Constructors





 ClientSideEvents Constructor	<p>The default public constructor for the ClientSideEvents object. This object is created automatically by the UltraWebGrid control.</p>
---	--

Public Properties










 AfterCellUpdateHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after a cell has been updated on the client.</p>
 AfterColumnMoveHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after a column has been moved on the client.</p>
 AfterColumnSizeChangeHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after the width of a column has changed on the client.</p>
 AfterEnterEditModeHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after the user begins editing a cell.</p>
 AfterExitEditModeHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after the value of a cell has been edited on the client.</p>
 AfterRowActivateHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after a row is activated on the client.</p>
 AfterRowCollapsedHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after a row is collapsed on the client.</p>
 AfterRowDeletedHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after a row is deleted on the client.</p>
 AfterRowExpandedHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after a row is expanded on the client.</p>
 AfterRowInsertHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after a row is added on the client.</p>
 AfterRowSizeChangeHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after the height of a row has changed on the client.</p>
 AfterRowTemplateCloseHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after a row edit template is closed on the client.</p>
 AfterRowTemplateOpenHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after a row edit template is opened on the client.</p>
 AfterSelectChangeHandler	<p>Returns or sets a string of text that specifies the name of the JavaScript function to be called after selection has changed on the client.</p>

 AfterSortColumnHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called after a column has been sorted on the client.
 BeforeCellChangeHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a row is activated on the client.
 BeforeCellUpdateHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a cell is updated on the client.
 BeforeColumnMoveHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a column is moved on the client.
 BeforeColumnSizeChangeHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before the width of a column is changed on the client.
 BeforeEnterEditModeHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a cell is edited on the client.
 BeforeExitEditModeHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before the value of the cell being edited is accepted on the client.
 BeforeRowActivateHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a row is activated on the client.
 BeforeRowCollapsedHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a row is collapsed on the client.
 BeforeRowDeletedHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a row is deleted on the client.
 BeforeRowExpandedHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a row is expanded on the client.
 BeforeRowInsertHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a row is added on the client.
 BeforeRowSizeChangeHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before the height of a row is changed on the client.
 BeforeRowTemplateCloseHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a row edit template is closed on the client.
 BeforeRowTemplateOpenHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a row edit template is opened on the client.
 BeforeSelectChangeHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before selection has changed on the client.
 BeforeSortColumnHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called before a column has been sorted on the client.
 CellChangeHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the active cell is changed on the client.


 CellClickHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the mouse's button is clicked over a cell of the grid on the client.
 ClickCellButtonHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when a cell's button is clicked on the client.
 ColumnDragHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when a column is being dragged on the client.
 ColumnHeaderClickHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the mouse is clicked over a column header of the grid on the client.
 DbClickHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when a cell is double-clicked on the client.
 EditKeyDownHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when a key is pressed during edit mode on the client.
 EditKeyUpHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when a key is released during edit mode on the client.
 GridCornerImageClickHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the left mouse button is clicked on the GridCornerImage image of the grid on the client.
 HasChanges (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 InitializeLayoutHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the internal document object model is being initialized on the client.
 InitializeRowHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when a row is being initialized on the client.
 IsTrackingViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 KeyDownHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when a key is pressed when the grid is active on the client.
 KeyUpHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when a key is released when the grid is active on the client.
 MouseDownHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the left mouse button is pressed over the grid on the client.
 MouseOutHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the cursor leaves an item of the grid on the client.
 MouseOverHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the cursor enters an item of the grid on the client.
 MouseUpHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the left mouse button is released over the grid on the client.

 RowSelectorClickHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the mouse is clicked over a row selector of the grid on the client.
 TemplateUpdateCellsHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the edit template is about to be closed on the client. The event is called for each control in the template that has the columnKey attribute set. To update cells values properly a developer has to handle the event.
 TemplateUpdateControlsHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the edit template is shown on the client. The event is called for each control in the template that has the columnKey attribute set. To initialize those controls values properly a developer has to handle the event.
 ValueListSelChangeHandler	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the value list selection is changed on the client.

Public Methods

 Commit (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 CopyFrom	Copies the ClientSideEvents object
 CreateBackup (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 LoadViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 Reset (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 Rollback (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 SaveViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 ToString	Overridden. Returns a string representation of an ClientSideEvents object.
 TrackViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	

Protected Properties

 ViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
--	--

See Also

[ClientSideEvents Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

Trigger Postback Within a Client-Side Event

It may be necessary to trigger a postback from within a client-side event. A `needPostBack` function is provided for this functionality. A `cancelPostBack` function is also provided.

To force a post-back from within a client side event handler call the `igtbl_needPostBack(gridName)` function where *gridName* is the ID of the grid you are working with. To cancel a post-back which is caused by a server side event handler from within a client side event handler call the `igtbl_cancelPostBack(gridName)` function where *gridName* is the ID of the grid you are working with.

```
igtbl_needPostBack(gridName);
```

```
igtbl_cancelPostBack(gridName);
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #3**.

Return A Band Using a Row ID

There are times when you may want to know what band a row is in. For example, you might want different actions to be performed in the **AfterRowActivate** event depending on what band a row is in. The UltraWebGrid provides the utility function "igtbl_getBandById" to accomplish this on the client-side.

This project creates a "Customers" DataTable and an "Orders" DataTable and adds them to a DataSet. Then a relationship between the two DataTables is created and added to the DataSet. The completed DataSet is bound to the UltraWebGrid. Finally, a client-side event handler is added to handle the **AfterRowActivate** event.

To obtain the Visual Studio.NET project files associated with this topic, [click this link](#). (This link points to a resource on the Infragistics website and requires an active Internet connection.)

The project consists of the following Classes:

clsCustomerData

The clsCustomerData Class provides a method for the programmatic generation of the Customers DataTable. This code is not reviewed.

clsOrderData

The clsOrderData Class provides a method for the programmatic generation of the Orders DataTable. This code is not reviewed.

WebForm1.aspx.vb

The WebForm1 Class contains a **Page_Load** event procedure. This event contains the code relevant to this project. You may recognize the bulk of the code from "Binding to a DataSet". The one new line is indicated as such.

1. Declare a New DataSet and name it "CustomerOrders":

In Visual Basic:

```
' declare DataSet to contain Hierarchical data  
Dim dataset As System.Data.DataSet = New System.Data.DataSet("CustomerOrders")
```

2. Create a DataTable containing Customer data and add it to the DataSet:

In Visual Basic:

```
' make Customers DataTable  
Dim custData As New clsCustomerData()  
dataset.Tables.Add(custData.MakeDataTable)
```

3. Create a DataTable containing Order data and add it to the DataSet:

In Visual Basic:

```
' make Orders DataTable  
Dim ordData As New clsOrderData()  
dataset.Tables.Add(ordData.MakeDataTable(dataset.Tables("Customers")))
```

4. Create a relationship between the Customers and Orders and add it to the DataSet:

In Visual Basic:

```
' create customers/orders relationship and add to DataSet
Dim relCustOrder As New DataRelation("CustOrder", dataset.Tables("Customers").Columns
("CustomerID"), dataset.Tables("Orders").Columns("CustomerID"))
dataset.Relations.Add(relCustOrder)

' set the ultrawebgrid's view type to Hierarchical
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.
Hierarchical
```

5. Create a client-side handler for activating a row

In Visual Basic:

```
' add the client-side event handler
'(The line below is the only new line from "Binding to a DataSet")
UltraWebGrid1.DisplayLayout.ClientSideEvents.AfterRowActivateHandler = "AfterRowActivate"
```

6. Bind the DataSet to the UltraWebGrid:

In Visual Basic:

```
' bind the DataSet to the Grid
UltraWebGrid1.DataSource = dataset
UltraWebGrid1.DataBind()
```

WebForm1.aspx

The implementation of the AfterRowActivate event handler must now be added to the ASPX page. The following JavaScript code is used to create a message box displaying the key for the band. In the Design View, click the button marked "HTML" at the bottom of the designer to see the underlying HTML.

```
function AfterRowActivate(tableName, rowId)
{
    // Get the band object based on the passed-in row ID.
    var band = igtbl_getBandById(rowId);

    // Display the band's key in a message box.
    alert(band.Key);

    // Do something specific based on the band's key
    if(band.Key == "Customers")
    {
        alert("Do something here for the parent band");
    }
    else if (band.Key == "Orders")
    {
        alert("Do something here for the child band");
    }
}
```

Run the sample. Click a row selector to activate a row. (The row selector is located between the first column of data and the expansion indicator ("+" sign). Since this grid has no formatting set, this may not be readily apparent.) Once you have successfully clicked the row selector, the row will be activated (as indicated by the dotted line around the row), and a message box will display the key for the band. For the parent band the value displayed should be "Customers". For the child band the value displayed should be "Orders". A second message box will then show. The text of this message box is determined by the value of the band's key.

See the documentation for the client-side Band object for additional properties that may be queried or set.

Task-Based_Help_Topic

Looping or iterating through the object collections in the client-side object model is a commonly required task. The code samples in this article demonstrates some common techniques.

This example iterates the rows of band 0 and expands each one so that the rows of band 1 are visible. For each row in band 0, any child rows are then selected.

```
function ExpandAllRows() {
    // Loop thru the rows of Band 0 and expand each one
    // Use the variable automatically declared for the grid on the page.
    var oGrid = oUltraWebGrid1;
    var oRows = oGrid.Rows;
    for(i=0; i<oRows.length; i++) {
        oRow = oRows.getRow(i);
        oRow.setExpanded(true);
        if(oRow.ChildRowsCount >> 0) {
            var oChildRows = oRow.Rows;
            for(j=0; j<oChildRows.length; j++) {
                oChildRow = oChildRows.getRow(j);
                oChildRow.setSelected(true);
            }
        }
    }
}
```

This example counts the columns of each band. For one column, (Phone) the column is made non-editable.

```
function CountColumns(btnEl) {
    // Count all columns and place the value into the label field
    // Use the variable automatically declared for the grid on the page.
    var oGrid = oUltraWebGrid1;
    // Get the Bands collection
    var oBands = oGrid.Bands;
    // Get the columns collection for Band 0
    var oBand = oBands[0];
    var oColumns = oBand.Columns;
    // The column count is equal to the length of the Columns array.
    var count = oColumns.length;
    var total = 0;
    // Add up the number of columns in each band.
    for(i=0; i<oBands.length; i++) {
        oBand = oBands[i];
        total += oBand.Columns.length;
        // iterate the columns of the band
        for(c = 0; c < oBand.Columns.length; c++) {
            var column = oBand.Columns[c];
            if(column.Key == "Phone")
                column.AllowUpdate = 2; // AllowUpdate.No
        }
    }
}
```

This example counts the cells within the first 2 bands of the grid and visits each cell and sets a background color on the HTML Element style object.

```
function CountGridCells() {
```

```
var oGrid = oUltraWebGrid1;
var oRows = oGrid.Rows;
var total = 0;
for(i=0; i<oRows.length; i++) {
    oRow = oRows.getRow(i);
    total += oRow.Band.Columns.length;
    for(c = 0; c < oRow.Band.Columns.length; c++) {
        var cell = oRow.getCell(c);
        cell.Element.style.backgroundColor = "yellow";
    }
    var oChildRows = oRow.Rows;
    for(j=0; j<oChildRows.length; j++) {
        oChildRow = oChildRows.getRow(j);
        total += oChildRow.Band.Columns.length;
        for(c = 0; c < oChildRow.Band.Columns.length; c++) {
            var cell = oChildRow.getCell(c);
            cell.Element.style.backgroundColor = "green";
        }
    }
}
}
```

Suspend Data Updates on the Client-Side

Some operations on the client take longer than others because they involve updating tables that will be sent back to the server so that those changes will be reflected on subsequent postbacks. Row and Cell selection are one example. Cell edits and row expansion state are others. If a large number of such updates need to be performed all at once, then the `WebGrid.suspendUpdates(true)` method should be used to turn off the internal updating sequence until after all changes have been made. Once the updates are complete, `suspendUpdates(false)` must be called so that WebGrid can update the internal tables with the updated values.

This example shows the row selection process executing for all rows of a multiband grid. `suspendUpdates()` is called at the beginning of the process and again at the end to suspend internal updates and then resume them once again. This improves the performance of this operation considerably.

```
function SelectAllRows() {
    // Select all rows in the grid
    var oGrid = oUltraWebGrid1;
    var oRows = oGrid.Rows;
    oUltraWebGrid1.suspendUpdates(true);
    for(i=0; i<oRows.length; i++) {
        oRow = oRows.getRow(i);
        oRow.setSelected(true);
        if(oRow.ChildRowCount > 0) {
            var oChildRows = oRow.Rows;
            for(j=0; j<oChildRows.length; j++) {
                oChildRow = oChildRows.getRow(j);
                oChildRow.setSelected(true);
            }
        }
    }
    oUltraWebGrid1.suspendUpdates(false);
}
```

Delete Rows On Client-Side

In order to delete rows on the client in javascript you need to use the javascript function: `igtbl_deleteRow` (`gridName,rowID`). First, be sure to allow the deletion itself by setting the `AllowDeleteDefault` property in the `DisplayLayout` object or `AllowDelete` on the specific band.

The function needs to be passed the name of the grid and the ID of the row you want to delete.

The following sample Javascript code would delete all rows on the client in a function.

```
function delete() {
    //get the first row in the grid
    var row=igtbl_getRowById("UltraWebGridlr_0");
    //delete the first row in the grid
    igtbl_deleteRow("UltraWebGrid1","UltraWebGridlr_0");
    //create a counter for the row id
    var cnt=0;
    //create a loop, if the row has a next sibling then we need to delete it
    while(row.NextSibling!=null)
    {
        //increment the counter for the next rowID
        cnt+=1;
        //get the row current row using the name of the grid and the row number from our
counter so we can check it for a sibling
        row=igtbl_getRowById("UltraWebGridlr_"+cnt)
        //finally delete that row,
        igtbl_deleteRow("UltraWebGrid1","UltraWebGridlr_"+cnt);
    }
}
```

All the selected rows within the grid can be deleted as well. For deleting selected rows, use the following function: **`igtbl_deleteSelRows(gridName)`**.

```
igtbl_deleteSelRows("UltraWebGrid1");
```

Updating Cells On The Client

UltraWebGrid provides the ability to update cells on the client through JavaScript using the client-side object model. Any cell value can be changed on the client without the need for a server-side post-back. Cells that are changed on the client are automatically posted to the server on the next post-back.

The following code obtains the Row object for the cell that is exiting edit mode. Using the Row object it updates the cell in column 9 with a new value.

This code assumes that BeforeExitEditMode has been specified as the client-side event handler for the BeforeExitEditMode event.

JavaScript:

```
function BeforeExitEditMode(tableName, itemName) {
    // Obtain the Row object for the current cell
    var row = igtbl_getRowById(itemName);
    // Obtain the Cell object for Column 9
    var cell = row.getCell(9);
    // Update the Cell value in Column 9
    cell.setValue("Updated");
    return 0;
}
```


Adding Rows and The AddNew Box

The AddNewBox is the area that contains the button that is made available to the user for adding rows to a band in the UltraWebGrid. There is a button provided for each band or level in the hierarchy of the data contained in the UltraWebGrid. By default, the Hidden property of the AddNewBox is set to true; in order to view this area, you would need to set this property to false. In order to be able to add rows, the AllowAddNew property needs to be set to true as well. The appearance of the AddNewBox can be changed through the Style object of the AddNewBox.

The following code makes the AddNewBox visible, changes the setting for allowing the addition of rows and sets the BackColor of the AddNewBox:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.AddNewBox.Hidden = False
UltraWebGrid1.DisplayLayout.AllowAddNewDefault = Infragistics.WebUI.UltraWebGrid.AllowAddNew.
Yes
UltraWebGrid1.DisplayLayout.AddNewBox.Style.BackColor = Color.Red
```

In C#:

```
UltraWebGrid1.DisplayLayout.AddNewBox.Hidden = false;
UltraWebGrid1.DisplayLayout.AllowAddNewDefault = Infragistics.WebUI.UltraWebGrid.AllowAddNew.
Yes;
UltraWebGrid1.DisplayLayout.AddNewBox.Style.BackColor = Color.Red;
```

It is also possible to add rows to the grid through code. To add rows through the server-side, use the Add method of the Rows collection. The following code will add a new row to the very bottom of the grid:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.Rows.Add()
```

In C#:

```
UltraWebGrid1.DisplayLayout.Rows.Add();
```

If you would like to add rows through client-side script, the Hidden Property of the AddNewBox must be set to false. If you do not want the AddNewBox to be visible, but still want the capability to add rows on the client-side then use the CustomRules property of the AddNewBox to hide it. The following code will hide the AddNewBox through the CustomRules property:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.AddNewBox.Style.CustomRules = "display:none;"
```

In C#:

```
UltraWebGrid1.DisplayLayout.AddNewBox.Style.CustomRules = "display:none";
```

The JavaScript necessary to add a row is below. The AddNew method must be passed the ID of the grid and the index of the band in which to add the row:

```
igtbl_addNew("UltraWebGrid1",0);
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #2**.

Retrieve Hidden Cell Values on Client-Side

This article discusses how to retrieve values of hidden cells through JavaScript on the client-side.

The first step in having cell values available on the client-side would be to hide the column on the server-side through the Custom Rules property. The following code will hide the first column and its header.

Place this code within the **Page_Load** event:

In Visual Basic:

```
UltraWebGrid1.Bands(0).Columns(0).CellStyle.CustomRules = "display:none"  
UltraWebGrid1.Bands(0).Columns(0).HeaderStyle.CustomRules = "display:none"
```

The following JavaScript code will place the value of a hidden cell on a label when the DoubleClick function is called:

```
function DoubleClick(tableName, itemName)  
{  
    var row = igtbl_getRowById(itemName);  
    var namevalue1 = row.getCell(0).getValue();  
    var label = igtbl_getElementById("Label1");  
    label.innerHTML = namevalue1;  
}
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #3**.

Change Row Height on Client-Side

In the course of a web application, you may find it necessary to programmatically change the height of your rows. The UltraWebGrid provides no utility function to do this, so you must make use of some of the Document Object Model's properties.

The sample project increases the size of a row when double clicked. The code for this project is located in the folder marked "Change A Row Height"

To obtain the Visual Studio.NET project files associated with this topic, [click this link](#). (This link points to a resource on the Infragistics website and requires an active Internet connection.)

WebForm1.aspx.vb

This code creates the datasource for the grid, and sets colors and widths to a couple of elements. The last two lines are the only lines specific to this sample. These lines will be reviewed.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.RowStyleDefault.Height = Unit.Pixel(40)
UltraWebGrid1.DisplayLayout.ClientSideEvents.DblClickHandler = "DoubleClick"
```

The first line sets a default height to the rows. The second tells the grid that we want to handle the double click event with client-side script. Let's now examine the "DoubleClick" function in the HTML.

WebForm1.aspx

The script below defines two constants for use in the function. ROW_HEIGHT represents the initial height of a row, and must match the value set in the code-behind. ROW_HEIGHT_DELTA represents the amount by which the height of a row will change when it is double clicked.

The function "DoubleClick" takes two parameters. The first contains the name of the grid, e.g. "UltraWebGrid1". The second contains the name of the element double clicked. We get a row object from the element clicked, and verify that it is valid. If there is no pixelHeight set yet, we set it to our constant, ROW_HEIGHT, to match what we set on the server. Then, regardless, we increment it by our ROW_HEIGHT_DELTA.

```
<script>

ROW_HEIGHT = 40;
ROW_HEIGHT_DELTA = 10;

function DoubleClick(tableName, itemName){
    var row = igtbl_getRowById(itemName);
    if (row != null)
    {
        if (row.Element.style.height == "")
        {
            row.Element.style.height = ROW_HEIGHT;
        }
        oldheight = parseInt(row.Element.style.height.replace(/px/, ''));
        igtbl_resizeRow("UltraWebGrid1", row.Element.id, oldheight + ROW_HEIGHT_DELTA)
    }
}
</script>
```

Note The syntax "variablename += 1" is equivalent to "variablename = variablename + 1".

Run the project, and double click different rows in the grid. Watch the rows expand.

Change Column Width

You can change the width of columns in web grid at designtime or at runtime.

At Design-Time

1. Click on the Columns collection in the property pages.
2. Select the column you wish to change.
3. Entering an Integer will set the width in pixels.
4. Entering a % will set the width of the column to that percentage of the total size of the grid (example: 25% will set the column to be 25% of the total grid width).

At Run-Time

In Visual Basic:

```
UltraWebGrid1.Columns(0).Width = Unit.Pixel(100)  
UltraWebGrid1.Columns(0).Width = Unit.Percentage(33.3)
```

In C#:

```
UltraWebGrid1.Columns(0).Width = Unit.Pixel(100);  
UltraWebGrid1.Columns(0).Width = Unit.Percentage(33.3);
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #2**.

Change Header Text on Client-Side

In the course of a web application, you may find it necessary to programmatically change the text of a column header. The UltraWebGrid provides no utility function to do this, so you must make use of the Document Object Model's **innerText** property.

1. Open a project (or a copy of a project) that contains an UltraWebGrid.
2. Add an HTML button in the design view, switch to the HTML view, and specify the "onclick" attribute to be a javascript function named "changeHeader()".

The HTML for your button will look similar to this:

```
<INPUT onclick="changeHeader();" style="Z-INDEX: 102; LEFT: 552px; POSITION: absolute; TOP: 27px" type="button" value="Button">
```

3. At the top of your HTML, within the <HEAD> tags, add this script block:

```
<script>
function changeHeader(){
    var col = igtbl_getElementById("UltraWebGrid1c_0_2");
    col.innerText = "New Text";
}
</script>
```

What this does is get the HTML element for the column header and sets the text of it to "New Text." Note that the format of the id for a column header is this: "GridName" + "C_" + Band Number + "_" + Column number. So the example shown here will change the third (index = 2) column's header.

4. Run the project, and click the button. Watch the header text change.

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #2**.

Change Grid Formatting Through Code

In the course of a web application, you may find it helpful to the user to change colors or fonts as a result of some grid event.

To obtain the Visual Studio.NET project files associated with this topic, [click this link](#). (This link points to a resource on the Infragistics website and requires an active Internet connection.)

clsCustomerData.vb

This code creates a datatable with several rows of customers. This code is not reviewed here.

WebForm1.aspx.vb

In the **Page_Load** event, the datatable is created and bound to the WebGrid. This event is not reviewed. The InitializeLayout event contains the following code:

In Visual Basic:

```
' Increase the width of the 2nd and 3rd columns (indexes 1 and 2)
UltraWebGrid1.DisplayLayout.Bands(0).Columns(1).Width = Unit.Pixel(200)
UltraWebGrid1.DisplayLayout.Bands(0).Columns(2).Width = Unit.Pixel(200)

' Set some colors that will be changed later on the client
UltraWebGrid1.DisplayLayout.HeaderStyleDefault.BackColor = Color.CadetBlue
UltraWebGrid1.DisplayLayout.RowStyleDefault.BackColor = Color.AliceBlue
UltraWebGrid1.DisplayLayout.RowAlternateStyleDefault.BackColor = Color.BlanchedAlmond

' Register the client-side event handler
UltraWebGrid1.DisplayLayout.ClientSideEvents.DblClickHandler = "DoubleClick"
```

With the comments, this code is self-explanatory. The last line may be tricky though; in order to use a specific client-side event, you must tell the grid that you will be doing so. The last line above shows one way of doing this. Alternatively, this property can be set in the property sheet at design-time.

WebForm1.aspx

Switch to the design view. Click the HTML view button in the lower left corner of the screen. You will see three functions in the code. Review them one at a time.

Double-click is the event you registered with the grid in the server side code. It takes 2 parameters, the *tableName* and the *itemName* of the part of the grid double clicked. Here the cell that was clicked on is returned by the **igtbl_getCellById** method, and if it is a cell, the *fontWeight* is set to "Bold". By querying *cell.Element*, you are accessing the HTML <TD> element. The *style.fontWeight* property is a standard Cascading Style Sheets property.

Reminder JavaScript is case sensitive. `igtbl_getCellById(itemName)` will not work, nor will `igtbl_GetCellById`, etc.

```
function DoubleClick(tableName, itemName){
    var cell = igtbl_getCellById(itemName);
    if (cell != null)
        cell.Element.style.fontWeight = "Bold";
}
```

The next two functions are run by clicking the buttons on the page. Each of these buttons has an HTML element that looks similar to this:

```
<INPUT onclick="changeCell();" style="Z-INDEX: 102; LEFT: 552px; POSITION: absolute; TOP: 27px" type="button" value="Button">
```

examine what happens in each function. *ChangeCell* queries the grids active cell, and changes the background color to red. *igtbl_getActiveCell("GridName")* returns a cell object that represents the current cell. "backgroundColor" is another of the standard CSS properties. This function closely resembles the previous one.

```
function ChangeCell(){
    var activeCell = igtbl_getActiveCell("UltraWebGrid1");
    if (activeCell != null)
        activeCell.Element.style.backgroundColor = "Red";
}
```

The last function in the sample will change the fontFamily of all the alternating rows. This is standard JavaScript and Document Object Model manipulation. This code is reviewed line-by-line.

On line 2, a "for" loop is initiated to iterate through all the styleSheet sections of the webpage. Line 3 gets an array of the CSS rules. Because Microsoft and Netscape have different names for their rules collections, you must access them differently. If you are unfamiliar with the use of the ternary operator (?), see the note below the code. A nested loop begins on line 4 to iterate through the CSS rules (or classes). Lines 5 and 6 say that if the rule is for the class your grid calls ".UltraWebGrid1-AltRowClass" then reset the fontFamily of that class to "Arial".

```
1  function ChangeAlternates(){
2      for(i = 0; i < document.styleSheets.length; i++){
3          rules = document.all ? document.styleSheets[i].rules : document.styleSheets[i].
cssRules;
4          for (j = 0; j < rules.length; j++){
5              if (rules[j].selectorText == ".UltraWebGrid1-AltRowClass"){
6                  rules[j].style.fontFamily = "Arial";
7              }
8          }
9      }
10 }
```

Line 3 could also have been written out in multiple steps. If this makes more sense, feel free to replace that line with this code:

```
if (document.all){
    // Only Microsoft browsers recognize "document.all"
    rules = document.styleSheets[i].rules;
}
else{
    // Netscape browsers will not.
    rules = document.styleSheets[i].cssRules;
}
```

Note When implementing in your own project, you will need to change the ".UltraWebGrid1-AltRowClass" to reflect the real name of the class you wish to change. If the grid were named "grdDetails" then your alternate row class would be called ".grdDetails-AltRowClass". Any time you set style properties in the grid, a CSS class is created. Check the HTML that is generated when you run the project for the names of these CSS classes.

Run the project. Select a cell in the grid. Click the buttons. Watch the color, font family and font weight (the "boldness") change as you click the buttons, or double click on a cell.

For more events, see the Sample that ships with the product called "ClientEvents".

Change the Color of the Grid's Scrollbar

You can change the color of the grid's scrollbar to match the overall look of your application. The following code illustrates how to change the scrollbar to blue. Enter the following code in the **InitializeLayout** event handler on the client side:

```
var d = igtbl_getElementById(gridName + "_div");  
d.style.scrollbarBaseColor="blue";
```


Activate Edit Mode Through Code

It may be necessary to provide the functionality of entering edit mode on a cell programmatically. The EnterEditMode function is provided for this task.

To enter edit mode, call the igtbl_EnterEditMode(gridName) function where the gridName is the ID of the grid. Calling this function will place the active cell in edit mode. The EnterEditMode function is only useful when a cell is the currently active cell in the grid. Also, this function is available only when activation is allowed.

The following JavaScript code will set a cell as the active cell and place that cell in edit mode:

```
//UltraWebGrid1 is the id of the WebGrid in use
var grid = igtbl_getGridById("UltraWebGrid1");
var row = igtbl_getRowById("UltraWebGrid1_0_0_0_0");
var cell = row.getCell[0];

//sets the active cell to the first cell in the grid
igtbl_setActiveCell("UltraWebGrid1", cell);
//places the cell in edit mode
igtbl_EnterEditMode("UltraWebGrid1");
```

Selectively Cancelling Keystrokes

There are times when it may be convenient to restrict the user's input to certain characters. For example, in a ZIP code field, you may only want to allow the user to enter numbers.

To obtain the Visual Studio.NET project files associated with this topic, [click this link](#). (This link points to a resource on the Infragistics website and requires an active Internet connection.)

WebForm1.aspx.vb

In the form load event a datatable with customer information is created and loaded. This code is not reviewed.

In the **InitializeLayout** event, there is one line:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ClientSideEvents.AfterEnterEditModeHandler = "AfterEnterEdit"
```

This line tells the grid that you have a client-side function called "AfterEnterEdit" that handles the **AfterEnterEditMode** client side event. If we were to call the event "DoSomething" then we would set the **AfterEnterEditModeHandler** to "DoSomething". You can name your client side events anything you want as long as you tell the grid what you named it.

WebForm1.aspx

In the .aspx page, click the HTML view button in the lower left corner of the Designer. Notice in the <HEAD> there is the following JavaScript block:

```
function AfterEnterEdit(tableName, cellId){
    var cell = igtbl_getCellById(cellId);

    var tb = igtbl_getElementById("UltraWebGrid1_tb");
    if (cell != null && cell.Column.Key == "CustomerID")
    {
        tb.onkeydown = onlyNumbersKeyHandler;
    }
    else
    {
        tb.onkeydown = null;
    }
}

function onlyNumbersKeyHandler(e){

    keycode = document.all ? event.keyCode : e.which;

    if ((keycode >= 48 && keycode < 57 /*0-9*/ ) ||
        (keycode >= 37 && keycode <= 40 /*Arrow keys*/ ) ||
        keycode == 8 /*backspace*/ ||
        keycode == 9 /*tab*/ || keycode == 27 /*escape*/ ||
        keycode == 13 /*enter*/ )
        return true; // allow these keystrokes

    return false; // disallow all others
}
```

This is where the work gets done. This may look strange if you have never worked with JavaScript or our client-side events before.

In the first function, we get the cell based on the cellID that was passed into the function. You are also getting a reference to the text box used for editing. The textbox has the id "GRIDNAME_tb", or "UltraWebGrid1_tb" in this sample. The code then checks to make sure that the cell is in the CustomerID column, and if so, sets the **onkeydown** event of the text box to the second function. Otherwise, it sets the **onkeydown** to null, so that the textbox's key down event is not handled at all.

The second function gets called when a key is pressed in the editing textbox. The variable "keycode" gets the ASCII value of the key that was pressed. This first line could also have been written like this:

```
if (document.all)
{
    // only IE recognizes document.all
    keycode = event.keyCode;
}
else
{
    // netscape doesn't
    keycode = e.which;
}
```

But using the "ternary operator" is more concise. The function then checks the keycode and returns true to allow certain characters and returns false to disallow all other characters.

Run the project. Double-click in the first column, and type. Notice that it only allows the number keys. Double Click in the second or third column, and notice that all keystrokes are allowed.

Change Cell Value in Response to Edit in Another Cell

It may be necessary to change the value of a cell in a row after the user has changed the value of a different cell within that row. This would be useful if one cell was based on a calculation of other cells in the row. First you would need a reference to the cell.

You can retrieve the cell within the **AfterExitEditMode** client-side handler with the following JavaScript:

```
var cell = igtbl_getCellById(itemName);
```

After you have a reference to the cell, you can get a reference to any other cell within that row and use the `setValue` method to change its value. The following script will change the value of the second cell in the row:

```
cell.Row.getCell(1).setValue("Test");
```

The complete event would be coded as:

```
function AfterExitEditMode(tableName, itemName)
{
    var cell = igtbl_getCellById(itemName);
    cell.Row.getCell(1).setValue("Test");
}
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #3**.

Expand and Collapse Rows on Client-Side

In order to expand or collapse rows on the client you need to call the toggleRow JavaScript Function.

Sample Code and Explanation

```
igtbl_toggleRow(gn,srcRow,expand)
```

Parameter descriptions:

gn= The name of your grid. **srcRow**= The ID of the row you want to expand or collapse. **expand**= true expands the row, false collapses it.

Examples (JavaScript)

To expand Row 0 of your UltraWebGrid:

```
igtbl_toggleRow('UltraWebGrid1','UltraWebGrid1r_0',true);
```

To collapse Row 0 of your UltraWebGrid:

```
igtbl_toggleRow('UltraWebGrid1','UltraWebGrid1r_0',false);
```

Prevent Row Deletion on Client-Side

The following topic demonstrates how to prevent a user from deleting a row through client-side script. The first step in preventing deletion is assigning a value to the **BeforeRowDeletedHandler** of the client-Side events. This property can be set either in the Property Pages under displaylayout -> ClientsideEvents -> BeforeRowDeletedHandler or in code:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ClientsideEvents.BeforeRowDeletedHandler = "BeforeDelete"
```

In C#:

```
UltraWebGrid1.DisplayLayout.ClientsideEvents.BeforeRowDeletedHandler = "BeforeDelete";
```

The handler receives 2 parameters, gridname and rowid. You would return true to cancel the event which would stop the delete. The following JavaScript demonstrates how to cancel the event for any row whose first cell has a value of true:

```
function BeforeDelete(gridname, rowid)
{
    //get the row that is about to be deleted
    var row=igtbl_getRowById(rowid);
    //get the value for column 1 in the current row
    var value=row.getCell(1).getValue();
    //check if the value is true, if so then don't delete the column
    //else don't do anything and the column will delete
    if(value==true)
    {
        return true;
    }
    //or in this case since deletion is based on the boolean value
    //you can just return the value and omit the if statement.
}
```

Handling Edit Mode on the Client-Side

This topic describes how to use the **EnterEditMode** and the **EndEditMode** functions of UltraWebGrid.

In the following functions, the gn parameter is the name of the grid.

To trigger edit mode, use:

```
function igtbl_EnterEditMode(gn) - Starts editing of the active cell.
```

To end edit mode, use:

```
function igtbl_EndEditMode(gn) - Ends editing.
```

Note that the functions are defined only if the activation is allowed. For example you can go right into the edit mode after a cell activating if you know that its value has to be changed:

```
function CellChange(gn, cellID)
{
var cell = igtbl_getCellById(cellID);
if(cell.getValue()>10)
    igtbl_EnterEditMode(gn);
}
```

Confirm Data Deletion With Message Box

The following will confirm a delete before deleting a row or several rows. This code will only ask the user to confirm the deletion once, regardless of the number of rows that are being deleted.

Add client side events for **KeyDown**, **BeforeDelete** and **AfterDelete**:

```
// needed to save a reference to
// the user's answer in the confirm box
var del=true;

function keyDown(grid, cellid, keycode)
{
    // if user hits delete key set deleting
    // to false so confirm box is shown
    if(keycode == 46)
        deleting = false;
}

function BeforeDelete(gridname, rowid)
{
    if(!deleting) // used to determine if confirm box has shown yet
    {
        // show the confirm box
        del = confirm("You sure you want to delete row(s)");
        // set deleting to true so confirm is not shown again
        deleting=true
    }
    // returning true cancels the event so return the opposite
    // of what the user selected in the confirm box
    return !del;
}

function AfterDelete()
{
    // set deleting to true so when deleting
    // multiple rows confirm is shown only once
    deleting=true;
    del=true;
}
```


Client-Side Searching (Find/Find Next)

WebGrid supports searching on the client as well as on the server. This example shows how to set up a Find - FindNext interface for locating and selecting cells with a particular value. The example assumes that there is a button with id "Find" and a text input with id "FindVal" on the same page as the grid. Once an initial Find is performed, the button text becomes "Find Next" and the next click performs a WebGrid.findNext() operation.

The HTML for the button and the text input control are shown first. The 'onkeydown' event calls the resetFind function to indicate a fresh search will be performed.

find() and findNext() are also support within a Band and within a Column object.

```
<input type="button" id="Find" onclick="javascript:FindValue(this);" style="Z-INDEX: 108; LEFT: 191px; WIDTH: 65px; POSITION: absolute; TOP: 56px; HEIGHT: 24px" Value="Find"> <input type="text" name="FindVal" id="FindVal" onkeydown="javascript:resetFind(this);" style="Z-INDEX: 105; LEFT: 32px; POSITION: absolute; TOP: 57px">
```

```
function resetFind() {
    var btnEl = igtbl_getElementById("Find");
    btnEl.value="Find";
}

function FindValue(btnEl) {
    var eVal = igtbl_getElementById("FindVal");
    findValue = eVal.value;
    var re = new RegExp("^" + findValue, "gi");
    // Determine if this is an initial find or a find next.
    if(btnEl.value=="Find") {
        igtbl_clearSelectionAll(oUltraWebGrid1.Id)
        var oCell = oUltraWebGrid1.find(re);
        if(oCell != null) {
            btnEl.value="Find Next";
            var row = oCell.Row.ParentRow;
            while(row != null) {
                row.setExpanded(true);
                row = row.ParentRow;
            }
            oCell.setSelected(true);
        }
    }
    else { // Otherwise it's a Find Next operation
        var oCell = oUltraWebGrid1.findNext();
        if(oCell == null) {
            btnEl.value="Find";
        }
        else {
            var row = oCell.Row.ParentRow;
            while(row != null) {
                row.setExpanded(true);
                row = row.ParentRow;
            }
            oCell.setSelected(true);
        }
    }
}
```

Popup a WebMenu on Right-Click Over Cell

Note This topic requires an existing installation of Infragistics UltraWebNavigator on your machine. UltraWebNavigator is a separate product that is not part of UltraWebGrid. UltraWebGrid and UltraWebNavigator are both part of the NetAdvantage suite.

For more information, visit our website at www.infragistics.com

By setting a CellClickHandler and checking for the right-mouse button, it is easy to display a popup menu within the grid.

1. Add and configure a WebGrid and a Popup WebMenu on the aspx page. The name of the WebMenu control is assumed to be 'UltraWebMenu1'.
2. In the ClientSideEvents object of WebGrid, add a CellClickHandler function named: GridCellClick
3. Define the GridCellClick function in the aspx page as follows:

```
<script language=javascript>
function GridCellClick (gridName,CellID,button) {
    if(button == 2) {
        igmenu_showMenu('UltraWebMenu1', event);
        return true;
    }
}
</script>
```

The function first checks the value of the third parameter to see if it is the right mouse button, (2). (The left mouse button would be 0.) If so, then the menu is displayed. The handling of menu item selections would then be performed normally by the WebMenu control.

If menu selections are to be handled on the client, then the Client-Side **ItemClick** event of WebMenu can be used to process that user action.

Selecting Rows on Client-Side

There are times when you may want to programmatically select a row on the client-side. For example, if you are providing a context menu on a right click, you will want the click to select the row first. The UltraWebGrid provides the utility function "igtbl_selectRow" to accomplish this.

This project creates a simple two row grid and a button. To obtain the Visual Studio.NET project files associated with this topic, [click this link](#). (This link points to a resource on the Infragistics website and requires an active Internet connection.)

The project consists of the following Classes:

clsCustomerData

The clsCustomerData Class provides a method for the programmatic generation of the Customers DataTable. This code is not reviewed.

WebForm1.aspx.vb

The WebForm1 Class contains a Page Load event procedure, where the data is loaded and the UltraWebGrid bound.

1. Set the grid to allow row selection:

In Visual Basic:

```
' set select type to single (only one row at a time)
UltraWebGrid1.DisplayLayout.SelectTypeRowDefault = Infragistics.WebUI.UltraWebGrid.
SelectType.Single
' set the background color of a selected row, so we can see the change
UltraWebGrid1.DisplayLayout.SelectedRowStyleDefault.BackColor = Color.AliceBlue
```

2. Bind the data:

In Visual Basic:

```
' Bind grid to datatable
Dim cust As New clsCustomerData()
UltraWebGrid1.DataSource = cust.MakeDataTable()
UltraWebGrid1.DataBind()
```

WebForm1.aspx

The following HTML and JavaScript code is used to select the first row on a click of the button. In the Design View, click the button marked "HTML" at the bottom of the designer to see the underlying HTML.

```
<script>
function Click()
{
    // Parameters: Gridname, Row ID, selected, fire events
    igtbl_selectRow("UltraWebGrid1", "UltraWebGrid1r_0", true, true);
    igtbl_updatePostField("UltraWebGrid1");
}

```

This is the HTML that creates the button:

```
<INPUT onclick="Click();" style="Z-INDEX: 102; LEFT: 66px; POSITION: absolute; TOP: 319px"
type="button" value="Select First Row">
```

Row ID's are a concatenation of the GridName, "r_", and the row number. The first row, row 0, of a grid named "UltraWebGrid1" is thus "UltraWebGrid1r_0". Many of the events pass in a row ID or cell ID for you to use. Note that `igtbl_updatePostField(gridname)` must be called if you programmatically change the selected rows collection. If you do not care if the selection is maintained on a postback, you do not have to take this step. Run the sample. Click the button provided ("Select First Row"). The first row is highlighted.

Change Active Row On Client-Side

You can change the active row in client-side code. Use the following code:

```
igtbl_setActiveRow(GridName, RowObject.Element)
```

where GridName is the name of the UltraWebGrid, and RowObject is a row object. You can acquire a row object using:

```
igtbl_getRowById(CellID)
```

where CellID is the ID of a particular cell.

If you don't have a particular cell in mind, and just want a row, just make up the name of the cell. The format for a cell ID in the first band is simply:

```
GridName + "rc_" + row index + "_" + column index
```

Another band down, the format would be:

```
GridName + "rc_" + parent row index + "_" + row index + "_" + column
```

Change Column Sort on Client-Side By Forcing Postback

It may be necessary in client-side script to cause a postback in order to sort by a different column.

To cause a postback in order to sort a column you may use the following JavaScript function to cause the postback:

```
igtbl_doPostBack(gridName, sortArgs)
```

The *gridName* is the ID of the grid you are using and *sortArgs* is a string of a specific format that contains information about the column you want to sort by. The *sortArgs* string has to have the following format:

```
'Sort:columnPath:shiftKey'
```

The *Sort* is a mandatory, literal part of the format which indicates that you are sorting by a column; *columnPath* is the path of the column you want to sort by (example: 0_0 for column 0 in band 0); *shiftKey* is a flag which imitates the shift key pressing (could be true or false). This parameter is useful when the SortMulti header click is allowed, otherwise it may be omitted.

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #4**.

De-Select All Rows on Client-Side

You may want to allow your users to deselect all the rows in the grid via a button click or some other event. The UltraWebGrid has a utility function called "igtbl_clearSelectionAll" to accomplish this.

Open a project that has cell or row selection enabled. The "Clientside Select Rows" tutorial is a good place start if you don't have any other projects. Add an HTML button in the design view, switch to the HTML view, and specify the "onclick" attribute to be a javascript function named "deselectAll()". The HTML for your button will look similar to this:

```
<INPUT onclick="deselectAll();" style="Z-INDEX: 102; LEFT: 552px; POSITION: absolute; TOP: 27px" type="button" value="Button">
```

At the top of your HTML, within the <HEAD> tags, add this script block:

```
<script>

function deselectAll()
{
    // Function takes one parameter: Gridname
    igtbl_clearSelectionAll("UltraWebGrid1");
    igtbl_updatePostField("UltraWebGrid1");
}
</script>
```

Run the project. Select a row (or more than one, if your project allows it). Click the button, and watch the row (s) be deselected.

Note *igtbl_updatePostField(gridname)* must be called if you programmatically change the selected rows collection. If you do not care if the selection is maintained on a postback, you do not have to take this step.

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #4**.

Select Rows and Columns on Client-Side

It may be necessary to select rows, cells and columns through client-side script. After programmatically selecting an object, it can be scrolled into view as well. The following functions are javascript samples of selecting rows, cells, and columns. There is also a method for scrolling objects into view.

```
//get the activerow of the grid
var myrow=igtbl_getActiveRow("UltraWebGrid1");

//deselect the active row in the grid
//selectrow function takes grid name, element id, a boolean that determines
//if it should be selected or unselected
igtbl_selectRow("UltraWebGrid1", myrow.Element.id, false);

//set the activerow
//Row variable would be the row number
igtbl_setActiveRow("UltraWebGrid1",igtbl_getElementById("UltraWebGrid1r_"+Row));

//also select the activerow in the grid
igtbl_selectRow("UltraWebGrid1", "UltraWebGrid1r_"+Row, true);

//scroll row into view, scroll method can take any element, row cell, column.
igtbl_scrollToView("UltraWebGrid1",igtbl_getElementById("UltraWebGrid1r_"+Row));

//select a column
//takes gridname, column id and whether or true to select, false to deselect
//this function also takes an optional boolean parameter that determines
//whether or not you want to fire the event
igtbl_selectColumn(gn,column.id,false,)

//selecting a cell follows the same paramters as the select column function
//except it takes a cell id instead of a column id
igtbl_selectCell(gn,cell.id,false,);
```

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #4**.

Manually Drop Down the WebCombo

The dropdown area of WebCombo can be programmatically displayed on the client side using the following code. In this example, a button control is added to the page and a javascript handler is setup to call the WebCombo.setDropDown method.

The variable used to reference the WebCombo, (oWebCombo1) is automatically generated to the page based on the name of the control on the server.

In HTML:

```
<input type="button"
onclick="javascript:ShowDropDown()"
value='Show DropDown'>
```

In JavaScript:

```
<script language=javascript>
function ShowDropDown() {
    oWebCombo1.setDropDown(true);
}
</script>
```

WebCombo As DropDown List

The WebCombo control can be used as a dropdown selection editor inside of a cell. Here is the step-by-step procedure for implementing this behavior:

1. With WebGrid on a form, add a WebCombo control to the form.
2. The **BorderStyle** of the WebCombo should be set to None for an improved appearance within WebGrid cells.
3. The WebCombo control can be configured and bound to data in any of the normal ways available to that control. A typical way of utilizing a cell dropdown is to display a human readable version of a column that is a foreign key of another table. In our example we will use the Products and Categories tables of the NWind.mdb database that installs with the product. For a complete working example of the code described in this article, see the ValueLists sample, second page: WebGrid with WebCombo. ValueList.
4. Add a database connection for the NWind database. Add OleDbDataAdapters for the Products and Categories tables. Create a dataset to contain the two tables.
5. Connect the WebGrid **DataSource** property to dataSet11 and the **DataMember** to the Products table. Edit the WebGrid columns for the Products table so that the **HeaderText** property for the CategoryID column reads CategoryName, as this is what the effect will be once WebCombo is attached to this column.
6. Connect the WebCombo DataSource property to dataSet11 and the DataMember to the Categories table. Edit the WebCombo columns collection for the Categories table so that the CategoryID column **Hidden** property is set to True. We don't need to see this column at all since the CategoryName column is what will be used to select the value.
7. Set the WebCombo **DataTextField** to CategoryName. This tells WebCombo which column to place in the top portion when a row is selected from the dropdown.
8. Set the WebCombo **DataValueField** to CategoryID. This tells WebCombo which column to use as the actual value of the control. The **DataTextField** and the **DataValueField** are used together by WebGrid to display the first and update the second as part of database processing.
9. In the code behind file, add the following code to make the connection between WebGrid and WebCombo:

In Visual Basic:

```
UltraWebGrid1.Columns.FromKey("CategoryID").AllowUpdate=AllowUpdate.Yes  
UltraWebGrid1.Columns.FromKey("CategoryID").Type=ColumnType.DropDownList  
UltraWebGrid1.Columns.FromKey("CategoryID").ValueList.WebCombo = WebCombo1
```

In C#:

```
UltraWebGrid1.Columns.FromKey("CategoryID").AllowUpdate=AllowUpdate.Yes;  
UltraWebGrid1.Columns.FromKey("CategoryID").Type=ColumnType.DropDownList;  
UltraWebGrid1.Columns.FromKey("CategoryID").ValueList.WebCombo = WebCombo2;
```

Change JavaScript Files

The JavaScript files that provide the client side functionality can be changed and edited by developers to modify or enhance the behavior of the grid. (However, Infragistics provides only limited support and assistance for this).

Note When making alterations to the JavaScript files it is important to change the file names so that conflicts and confusion do not arise. The JavaScript functionality is distributed among several files so that functionality that is not needed is not sent to the browser. Each filename contains the root name `ig_WebGrid`. All file names are keyed off of this root.

To change JavaScript files perform these steps:

1. Create a new root name. In this case we will use `ig_MyGrid` as the new root.
2. This implies the following new file names:
 - `ig_MyGrid.js`
 - `ig_MyGrid_an.js`
 - `ig_MyGrid_dl.js`
 - `ig_MyGrid_dom.js`
 - `ig_MyGrid_ft.js`
 - `ig_MyGrid_gb.js`
 - `ig_MyGrid_ie.js`
 - `ig_MyGrid_ie6.js`
 - `ig_MyGrid_kb.js`
 - `ig_MyGrid_ml.js`
 - `ig_MyGrid_nn.js`
 - `ig_MyGrid_ro.js`
 - `ig_MyGrid_srt.js`
3. Copy or rename all the existing JavaScript files to their respective files with the new name.
4. Change the `UltraWebGrid.DisplayLayout.JavaScriptFileName` property to `/ig_MyScripts/ig_MyGrid.js`

Using Tutorials

The Tutorials section provides guided learning through a series of illustrated exercises that show you how to use the control. There are two sub-sections. The **Introductory** sub-section provides a simple overview of just the main tasks involved with getting the WebGrid up and running in your application. It also includes a topic with links to additional code resources available from the Infragistics web site.

The **In Depth** sub-section provides a much more detailed and exhaustive set of exercises that cover most aspects of the product, from simple data binding to advanced client-side functionality.

Quick Start - Binding WebGrid To Flat Data

This Quick Start tutorial helps get UltraWebGrid bound to a flat dataset within a ASP.NET Web Application project. Before starting this tutorial the developer should be familiar with Visual Studio .NET. UltraWebGrid is a .NET element for the Visual Studio .NET environment. Information about Visual Studio .NET is available at msdn.microsoft.com. There are many other sources for documentation of ASP.NET, Visual Basic .NET and C# both printed and on-line.

The directions below describe the step-by-step process of adding the control to your form and setting it up to bind to a flat data source.

1. Launch VS.NET and start a new ASP.NET Web Application project. Project name is not important.

You can optionally add an Infragistics tab to your toolbox to contain all of your Infragistics controls. Click the heading below to see a description of how to do this.

Add a New Tab to the Toolbox

(Click this heading to see procedure.)

- a. Display Toolbox by clicking on the Toolbox Icon.
- b. Click the Pin symbol to have the Toolbox continue to display. Unpin it later when more screen real estate is needed for the forms designer or code editing.
- c. Right-Click on the toolbox and the context menu displays. Select Add Tab.
- d. Type "Infragistics" into the toolbox tab name.
- e. Right-Click on the Infragistics toolbox tab. Select Customize Toolbox...
- f. Select the .NET Framework Components tab. Note: This takes a while to display.
- g. Check all elements in the Infragistics Namespace.
- h. Click OK.

All of the Infragistics Web controls installed on your development system display and are enabled on the Infragistics tab.

Adding UltraWebGrid to a WebForm

2. Add an instance of UltraWebGrid to WebForm1. Design View should be selected for WebForm1.aspx
3. Click UltraWebGrid in the Toolbox then click on the Form Designer. Note: Drag and Drop can also place the UltraWebGrid on the Form Designer. Drag and Size the UltraWebGrid so it takes up most of the form.

Adding References

4. UltraWebGrid requires a reference to the Infragistics.WebUI.Shared Elements. *Check to be sure that this DLL has been added to the project's references and that its **CopyLocal** property has been set to **True**.* If you would like help on how to do this, click the heading below.

Adding the WebUI.Shared Reference

(Click this heading to see procedure.)

- a. Select Solution Explorer and expand the References node.
- b. Check the contents of the References node to see if the `Infragistics.WebUI.Shared` DLL is included in the Solution.
- c. If this DLL is not within the references, right-click on References and choose "Add Reference". Select

`Infragistics.WebUI.Shared` from the dialog that appears and click OK. `Infragistics.WebUI.Shared` will display under the References node.

- d. Select the `Infragistics.WebUI.Shared` reference in the Solution Explorer, right-click it, and choose "Properties".
- e. Check to ensure the **CopyLocal** property is set to True. If it is not, set it to True.

Retrieve Database Data Using Data Elements and Bind to Grid

5. Click the Data tab in the toolbox. Select the `OleDbDataAdapter` tool and double-click it.
6. The Data Adapter Configuration Wizard will begin. The first step would be to add a connection.
7. Select "New Connection..." The first screen of the Connection Wizard will appear.
8. Click the Provider tab and select the "MS Jet 4.0 OLEDB Provider" option from the list. Click the "Next" button. The Connection tab will become active.
9. On the Connection tab, you are prompted to select or enter a database name. Click the ellipsis (...) button to the right of the text box to browse for the database that you will use. Navigate to the directory where you installed UltraWebGrid and locate the `cars.mdb` file in the `samples\data` folder. When you have selected the file, click "Open" and the filename will appear on the Connection tab.
10. Click the "Test Connection" button to verify the connection. Once you see a message indicating that the connection was successful, you can click OK to complete the connection setup.
11. On the next screen of the wizard, you are prompted to choose a query type. Choose "Use SQL statements". Click the "Next" button to go to the next screen.
12. The following screen prompts you for the SQL statements to generate. You can enter SQL directly into the "What data should the data adapter load into the dataset?" text box. Type "SELECT * FROM Manufacturers" into the text box. Click the "Next" button when you are done.
13. View the wizard results. All items should be checked on this screen.
14. When you have verified the wizard settings, click "Finish". The `OleDbDataAdapter` control will then appear in the form's component tray.
15. Select the `OleDbDataAdapter` control in the form's component tray, right-click, and choose the option "Generate Dataset..." The Generate Dataset screen appears.
16. On the Generate DataSet screen, a default name is provided for the DataSet (`DataSet1`) and the "Manufacturers" table should already be checked in the list. Click OK to accept these defaults. `DataSet1` appears in the form's component tray. (You may want to check the name to make sure it is the one you specified.)
17. Select the `UltraWebGrid` on the form. In the property sheet, set the `DataSource` property to "`DataSet1`".
18. Set the `DataMember` property to "Manufacturers".
19. In the **Load** event of `WebForm1` enter the following code to fill the Dataset with data from the database and bind the `UltraWebGrid` to the dataset:

In Visual Basic:

```
OleDbDataAdapter1.Fill(DataSet11)
UltraWebGrid1.DataBind()
```

In C#:

```
oleDbDataAdapter1.Fill(dataSet11);
```

```
UltraWebGrid1.DataBind();
```

Remember, C# is case-sensitive. Make sure the capitalization is correct.

- 20.** Build and browse the application. The project will execute, and you will see the UltraWebGrid on the form filled with the data from the Manufacturers table.

You may also want to refer to the Flat Grid sample that ships with the product as another example binding to a flat dataset.

Quick Start - Binding WebGrid To Hierarchical Data

This Quick Start tutorial helps get UltraWebGrid bound to a hierarchical dataset within a ASP.NET Web Application project. Before starting this tutorial the developer should be familiar with Visual Studio.NET. UltraWebGrid is a .NET element for the Visual Studio .NET environment. Information about Visual Studio .NET is available at msdn.microsoft.com. There are many other sources for documentation of ASP.NET, Visual Basic .NET and C# both printed and on-line.

The directions below describe the step-by-step process of adding the control to your form and setting it up to bind to a flat data source.

1. Launch VS.NET and start a new ASP.NET Web Application project. Project name is not important.

You can optionally add an Infragistics tab to your toolbox to contain all of your Infragistics controls. Click the heading below to see a description of how to do this.

Add a New Tab to the Toolbox

(Click this heading to see procedure.)

- a. Display Toolbox by clicking on the Toolbox Icon.
- b. Click the Pin symbol to have the Toolbox continue to display. Unpin it later when more screen real estate is needed for the forms designer or code editing.
- c. Right-Click on the toolbox and the context menu displays. Select Add Tab.
- d. Type "Infragistics" into the toolbox tab name.
- e. Right-Click on the Infragistics toolbox tab. Select Customize Toolbox...
- f. Select the .NET Framework Components tab. Note: This takes a while to display.
- g. Check all elements in the Infragistics Namespace.
- h. Click OK.

All of the Infragistics Web controls installed on your development system display and are enabled on the Infragistics tab.

Adding UltraWebGrid to a WebForm

2. Add an instance of UltraWebGrid to WebForm1. Design View should be selected for WebForm1.aspx
3. Click UltraWebGrid in the Toolbox then click on the Form Designer. Note: Drag and Drop can also place the UltraWebGrid on the Form Designer. Drag and Size the UltraWebGrid so it takes up most of the form.

Adding References

4. UltraWebGrid requires a reference to the Infragistics.WebUI.Shared Elements. *Check to be sure that this DLL has been added to the project's references and that its **CopyLocal** property has been set to **True**.* If you would like help on how to do this, click the heading below.

Adding the WebUI.Shared Reference

(Click this heading to see procedure.)

- a. Select Solution Explorer and expand the References node.
- b. Check the contents of the References node to see if the `Infragistics.WebUI.Shared` DLL is included in the Solution.
- c. If this DLL is not within the references, right-click on References and choose "Add Reference". Select

`Infragistics.WebUI.Shared` from the dialog that appears and click OK. `Infragistics.WebUI.Shared` will display under the References node.

- d. Select the `Infragistics.WebUI.Shared` reference in the Solution Explorer, right-click it, and choose "Properties".
- e. Check to ensure the **CopyLocal** property is set to True. If it is not, set it to True.

Retrieve Database Data Using Data Elements and Bind to Grid

5. Click the Data tab in the toolbox. Select the OleDbDataAdapter tool and double-click it.
6. The Data Adapter Configuration Wizard will begin. The first step would be to add a connection.
7. Select "New Connection..." The first screen of the Connection Wizard will appear.
8. Click the Provider tab and select the "MS Jet 4.0 OLEDB Provider" option from the list. Click the "Next" button. The Connection tab will become active.
9. On the Connection tab, you are prompted to select or enter a database name. Click the ellipsis (...) button to the right of the text box to browse for the database that you will use. Navigate to the directory where you installed UltraWebGrid and locate the NWind.mdb file in the samples\data folder. When you have selected the file, click "Open" and the filename will appear on the Connection tab.
10. Click the "Test Connection" button to verify the connection. Once you see a message indicating that the connection was successful, you can click OK to complete the connection setup.
11. On the next screen of the wizard, you are prompted to choose a query type. Choose "Use SQL statements". Click the "Next" button to go to the next screen.
12. The following screen prompts you for the SQL statements to generate. You can enter SQL directly into the "What data should the data adapter load into the dataset?" text box. Type "SELECT * FROM Customers" into the text box. Click the "Next" button when you are done.
13. View the wizard results. All items should be checked on this screen.
14. When you have verified the wizard settings, click "Finish". The OleDbDataAdapter control will then appear in the form's component tray.
15. Select the OleDbDataAdapter control in the form's component tray, right-click, and choose the option "Generate Dataset..." The Generate Dataset screen appears.
16. On the Generate DataSet screen, a default name is provided for the DataSet (DataSet1) and the "Customers" table should already be checked in the list. Click OK to accept these defaults. DataSet1 appears in the form's component tray. (You may want to check the name to make sure it is the one you specified.)
17. Select the UltraWebGrid on the form. In the property sheet, set the DataSource property to "DataSet1"
18. Set the DataMember property to "Customers".

Adding Hierarchical Functionality

19. You have now defined a flat dataset. The UltraWebGrid will be populated with one band and no children. The first step in creating a hierarchical WebGrid is to set the **ViewType** property. The **ViewType** property determines the structure and functionality of the grid. The following code sets the **ViewType** property to hierarchical.

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.Hierarchical
```

In C#:

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.Hierarchical;
```

20. Now, add a table to the exiting dataset in code that will serve as the child table. We can use the same connection for this table since the data is coming from the NWind database for our new adapter. Our new adapter will bring in data from the Orders table. A relation needs to be added so the dataset knows which columns serve as the parent column and the child column.
21. In the **Page_Load** event of WebForm1 enter the following code to fill the Dataset with data from the database and bind the UltraWebGrid to the hierarchical dataset:

In Visual Basic:

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.Hierarchical
```

```
OleDbDataAdapter1.Fill(DataSet11)
```

```
Dim cmdOrders As New Data.OleDb.OleDbDataAdapter("Select CustomerID,OrderID,OrderDate, ShippedDate,ShipAddress,Freight from Orders", OleDbConnection1)  
cmdOrders.Fill(DataSet11, "Orders")  
DataSet11.Relations.Add("Orders", DataSet11.Tables("Customers").Columns("CustomerID"), DataSet11.Tables("Orders").Columns("CustomerID"))
```

```
UltraWebGrid1.DataSource = DataSet11.Tables("Customers").DefaultView  
UltraWebGrid1.DataBind()
```

In C#:

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.Hierarchical;
```

```
oleDbDataAdapter1.Fill(dataSet11);
```

```
System.Data.OleDb.OleDbDataAdapter cmdOrders = new System.Data.OleDb.OleDbDataAdapter ("Select CustomerID,OrderID,OrderDate,ShippedDate,ShipAddress,Freight from Orders", OleDbConnection1);  
cmdOrders.Fill(dataSet11, "Orders");  
dataSet11.Relations.Add("Orders", dataSet11.Tables["Customers"].Columns["CustomerID"], dataSet11.Tables["Orders"].Columns["CustomerID"]);
```

```
UltraWebGrid1.DataSource = dataSet11.Tables["Customers"].DefaultView;  
UltraWebGrid1.DataBind();
```

Remember C# is case-sensitive. Make sure the capitalization is correct.

22. Build and browse the application. The project will execute, and you will see the UltraWebGrid on the form filled with hierarchical data from the Northwind database.

Quick Start - Unbound WebGrid

This Quick Start tutorial helps get UltraWebGrid running in a ASP.NET Web Application project. Before starting this tutorial the developer should be familiar with Visual Studio .NET. UltraWebGrid is a .NET element for the Visual Studio .NET environment. Information about Visual Studio .NET is available at msdn.microsoft.com. There are many other sources for documentation of ASP.NET, Visual Basic .NET and C# both printed and on-line. The directions below describe the step-by-step process of adding the control to your form and setting it up.

1. Launch VS.NET and start a new ASP.NET Web Application project. Project name is not important.

You can optionally add an Infragistics tab to your toolbox to contain all of your Infragistics controls. Click the heading below to see a description of how to do this.

Add a New Tab to the Toolbox

(Click this heading to see procedure.)

- a. Display Toolbox by clicking on the Toolbox Icon.
- b. Click the Pin symbol to have the Toolbox continue to display. Unpin it later when more screen real estate is needed for the forms designer or code editing.
- c. Right-Click on the toolbox and the context menu displays. Select Add Tab.
- d. Type "Infragistics" into the toolbox tab name.
- e. Right-Click on the Infragistics toolbox tab. Select Customize Toolbox...
- f. Select the .NET Framework Components tab. Note: This takes a while to display.
- g. Check all elements in the Infragistics Namespace.
- h. Click OK.

All of the Infragistics Web controls installed on your development system display and are enabled on the Infragistics tab.

Adding UltraWebGrid to a WebForm

2. Add an instance of UltraWebGrid to WebForm1. Design View should be selected for WebForm1.aspx
3. Click UltraWebGrid in the Toolbox then click on the Form Designer. Note: Drag and Drop can also place the UltraWebGrid on the Form Designer. Drag and Size the UltraWebGrid so it takes up most of the form.

Adding References

4. UltraWebGrid requires a reference to the Infragistics.WebUI.Shared Elements. *Check to be sure that this DLL has been added to the project's references and that its **CopyLocal** property has been set to **True**.* If you would like help on how to do this, click the heading below.

Adding the WebUI.Shared Reference

(Click this heading to see procedure.)

- a. Select Solution Explorer and expand the References node.
- b. Check the contents of the References node to see if the `Infragistics.WebUI.Shared` DLL is included in the Solution.
- c. If this DLL is not within the references, right-click on References and choose "Add Reference". Select `Infragistics.WebUI.Shared` from the dialog that appears and click OK. `Infragistics.WebUI.Shared` will display under the References node.

- d. Select the `Infragistics.WebUI.Shared` reference in the Solution Explorer, right-click it, and choose "Properties".
- e. Check to ensure the **CopyLocal** property is set to True. If it is not, set it to True.

Adding Unbound Data

5. You can add unbound data through code. First you would have to add columns to the WebGrid. After adding columns, you may then add as many rows as you wish. Each cell can then be populated with values by accessing it through the Rows collection. The following code adds 3 columns and 100 rows to the WebGrid. In the third cell of the third row, the word "Hello" will be placed and within the second cell of the 89th row, "Row 88, Column 1" will be visible.

Place this code within the **Load** event of the page.

In Visual Basic:

```
Dim i As Int16
If Not Me.IsPostBack Then
    UltraWebGrid1.Columns.Add("First", "Column One")
    UltraWebGrid1.Columns.Add("Second", "Column Two")
    UltraWebGrid1.Columns.Add("Third", "Column Three")

    For i = 0 To 99
        UltraWebGrid1.Rows.Add(i.ToString())
    Next

    Dim row As Infragistics.WebUI.UltraWebGrid.UltraGridRow
    row = UltraWebGrid1.Rows.FromKey("2")

    Dim cell As Infragistics.WebUI.UltraWebGrid.UltraGridCell
    cell = row.Cells.FromKey("Third")
    cell.Value = "Hello"

    UltraWebGrid1.Rows(88).Cells(1).Value = "Row 88, Column 1"
End If
```

In C#:

```
if (!IsPostBack)
{
    UltraWebGrid1.Columns.Add("First", "Column One");
    UltraWebGrid1.Columns.Add("Second", "Column Two");
    UltraWebGrid1.Columns.Add("Third", "Column Three");

    for(int i = 0; i <= 99; i++) {
        UltraWebGrid1.Rows.Add(i.ToString());
    }
    UltraGridRow row = UltraWebGrid1.Rows.FromKey("2");
    UltraGridCell cell = row.Cells.FromKey("Third");
    cell.Value = "Hello";
    UltraWebGrid1.Rows[88].Cells[1].Value = "Row 88, Column 1";
}
```

Creating an Unbound Grid through the Property Pages

6. An unbound grid can also be created without writing a single line of code. First, follow the steps above detailing how to add the UltraWebGrid to a form. Be sure that the `Infragistics.WebUI.Shared` DLL exists in the Solution Explorer and its CopyLocal property is set to true.
 - a. Click on the WebGrid and proceed to its Properties Window.

- b. Click within the **Columns** property and you will notice an ellipsis. Click on the ellipsis to open up the Columns Collection Editor.
- c. Click on the Add button to add as many columns as needed.
- d. Notice as the columns are added their properties become available within the Columns Collection Editor as well.
- e. Change the **HeaderText** property to reflect the visible caption desired within the header.
- f. Be sure to click the Apply button to apply these settings within the UltraWebGrid.

Now that you have columns, rows can be added. Remember, **columns must be present before adding rows**.

- g. Within the Properties Window for the WebGrid, click on the **Rows** property and the ellipsis to open up the Rows Collection Editor.
- h. Add as many rows as needed.
- i. Notice as the rows are added, the properties for the rows are available. You could have a different **Style** for every row in the grid.

Now that you have rows, cells can be added.

- j. To add cells, click on the **Cells** property within the Rows Collection editor. Clicking on the ellipsis within that property will bring up the Cells Collection Editor.
- k. You can set the **Text** property of the cells as well as properties affecting the appearance of the Style object within the editor.
- l. Build and browse your page to see the structure and data you have created within the UltraWebGrid.

Be sure to review the Unbound Mode sample that ships with the product. Also, you may want to review the topic [Using the Grid in Unbound Mode](#).

There is an additional code sample that illustrates the concepts discussed in this topic. The [Additional Code Resources topic](#) has a link to this sample as well as several others. Look for **WebGrid Sample Project #3**.

Creating a WebGrid Programmatically

This topic shows how to dynamically create the UltraWebGrid on a WebForm within a ASP.NET Web Application project without having to place it there at design-time. Before starting this tutorial, the developer should be familiar with Visual Studio .NET. UltraWebGrid is a .NET element for the Visual Studio .NET environment. Information about Visual Studio .NET is available at msdn.microsoft.com. There are many other sources for documentation of ASP.NET, Visual Basic .NET and C# both printed and on-line.

Place the following code within the **Page_Load** event:

In Visual Basic:

```
Dim UltraWebGrid1 As Infragistics.WebUI.UltraWebGrid.UltraWebGrid

UltraWebGrid1 = New Infragistics.WebUI.UltraWebGrid.UltraWebGrid("UltraWebGrid1")
UltraWebGrid1.Columns.Add("col1", "Column 1")
UltraWebGrid1.Columns.Add("col2", "Column 2")
UltraWebGrid1.Columns.Add("col3", "Column 3")

Dim cells(3) As Object
cells(0) = "Cell11"
cells(1) = "Cell12"
cells(2) = "Cell13"

UltraWebGrid1.Rows.Add(New Infragistics.WebUI.UltraWebGrid.UltraGridRow(cells))
UltraWebGrid1.Rows.Add(New Infragistics.WebUI.UltraWebGrid.UltraGridRow(cells))
UltraWebGrid1.Rows.Add(New Infragistics.WebUI.UltraWebGrid.UltraGridRow(cells))
Me.Controls.Add(UltraWebGrid1)
```

In C#:

```
protected Infragistics.WebUI.UltraWebGrid.UltraWebGrid UltraWebGrid1;
UltraWebGrid1 = new Infragistics.WebUI.UltraWebGrid.UltraWebGrid("UltraWebGrid1");
UltraWebGrid1.Columns.Add("col1", "Column 1");
UltraWebGrid1.Columns.Add("col2", "Column 2");
UltraWebGrid1.Columns.Add("col3", "Column 3");

object [] cells = { "Cell11", "Cell12", "Cell13"};
UltraWebGrid1.Rows.Add(new UltraGridRow(cells) );
UltraWebGrid1.Rows.Add(new UltraGridRow(cells));
UltraWebGrid1.Rows.Add(new UltraGridRow(cells));
this.Controls.Add(UltraWebGrid1);
```

Additional Code Resources

In addition to the code reproduced in the help file and the topic-specific example projects available in some of the Task-Based Help topics, there are a number of example projects that illustrate important WebGrid concepts. The following list of samples provides a brief description of the code projects currently available. All the links in this topic point to resources on the Infragistics web site, and require an active Internet connection to work.

WebGrid Sample Project #1

This sample demonstrates how to perform several client side tasks: adding rows, deleting rows, confirming/canceling deletes, expanding rows, collapsing rows. In the HTML you will see many utility functions that are used to select rows, deselect rows, set activerow, get parent rows/child rows/sibling rows. Initializing row values on the server and expanding/collapsing all on the server.

[Click here for the knowledge base article on this sample, including a link to download the file\(s\).](#)

WebGrid Sample Project #2

This sample demonstrates a number of concepts including updating header and footer text, arrange columns in the desired order and setting their widths, hiding the addnew box but allowing addnew functionality, adding rows without default values when default values are set, data validation on both the client and the server, allowing the user to resize rows independently, and looping through all the rows in a given band.

[Click here for the knowledge base article on this sample, including a link to download the file\(s\).](#)

WebGrid Sample Project #3

This sample demonstrates how to set the ActiveCell as well as the functionality that can be used within the UltraWebGrid's client-side functions. Clicking on a cell button will fire a postback using JavaScript. Double-clicking on a row selector will retrieve the value of a hidden cell through client-side code as well. JavaScript is also used to keep the contents of the first row from being edited by the user. Pressing the button to the right will print the web page including the UltraWebGrid.

[Click here for the knowledge base article on this sample, including a link to download the file\(s\).](#)

WebGrid Sample Project #4

When the page first loads, the first row is the ActiveRow and the third row is selected. The RowSelector indicates the ActiveRow - there can only be 1 ActiveRow at any given time, while several rows can be selected at once. Different appearances can be set for the ActiveRow and selected rows as well. Also, this sample demonstrates how to use some additional client-side functionality with UltraWebGrid. You can sort a column, make selections and clear all selections through script. Scrolling can also be accomplished.

[Click here for the knowledge base article on this sample, including a link to download the file\(s\).](#)

Quick Start 1

There are 6 major steps to this exercise:

- [Background](#)
- [Solution](#)
- [Sample Project](#)
- [Code Discussion](#)
- [Review](#)

Background

This Quick Start tutorial helps get UltraWebGrid running in a VB.NET Web Application Project. Before starting this tutorial the developer should be familiar with Visual Studio .NET.

UltraWebGrid is a .NET element for the Visual Studio .NET environment. Information about Visual Studio .NET is available at msdn.microsoft.com. There are many sources of information on .NET, both printed and on-line.

UltraWebGrid is licensed as a part of the NetAdvantageSuite. Installation documentation is included with each product. Trial versions are available for download.

Solution

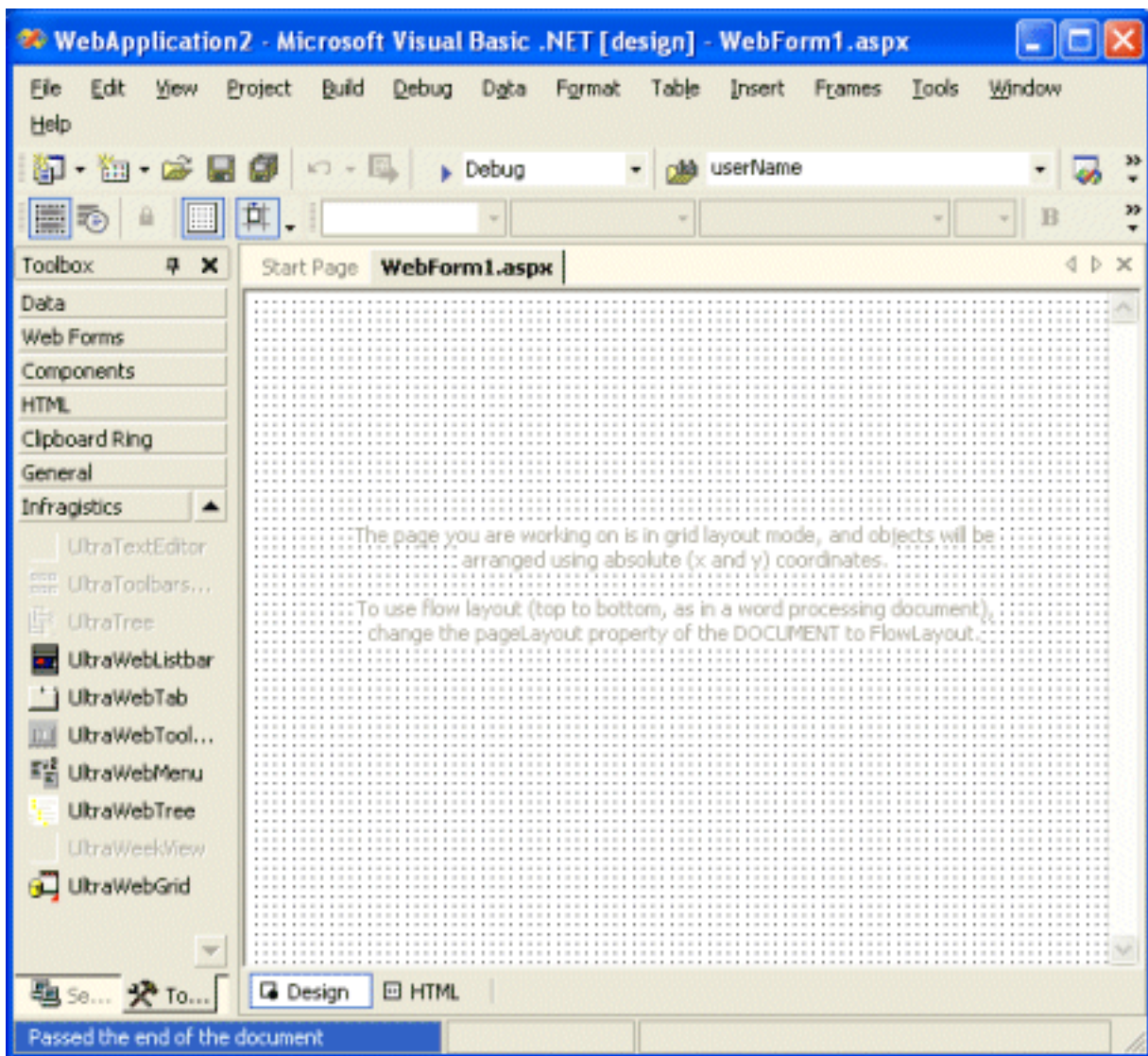
This tutorial walks the developer through the steps required to open a new project, place the UltraWebGrid on a Web Form, put some data in the grid and test the project.

Getting Started

Perform the following steps to start Visual Studio and add the Infragistics controls to the Toolbox:

1. From the Start menu, launch Microsoft Visual Studio .NET and start a new ASP.NET Web Application project. The project name is not important
2. Display Toolbox by clicking on the Toolbox Icon.
3. Click the Pin symbol to have the Toolbox continue to display. Unpin it later when more screen real estate is needed for the forms designer or code editing.
4. Right-Click on the toolbox and the context menu displays. Select Add Tab.
5. Type Infragistics into the toolbox tab name.
6. Right-Click on the Infragistics toolbox tab. Select Customize Toolbox
7. Select the .NET Framework Components tab. Note: This takes a while to display.
8. Check all elements in the Infragistics Namespace.
9. Click OK.

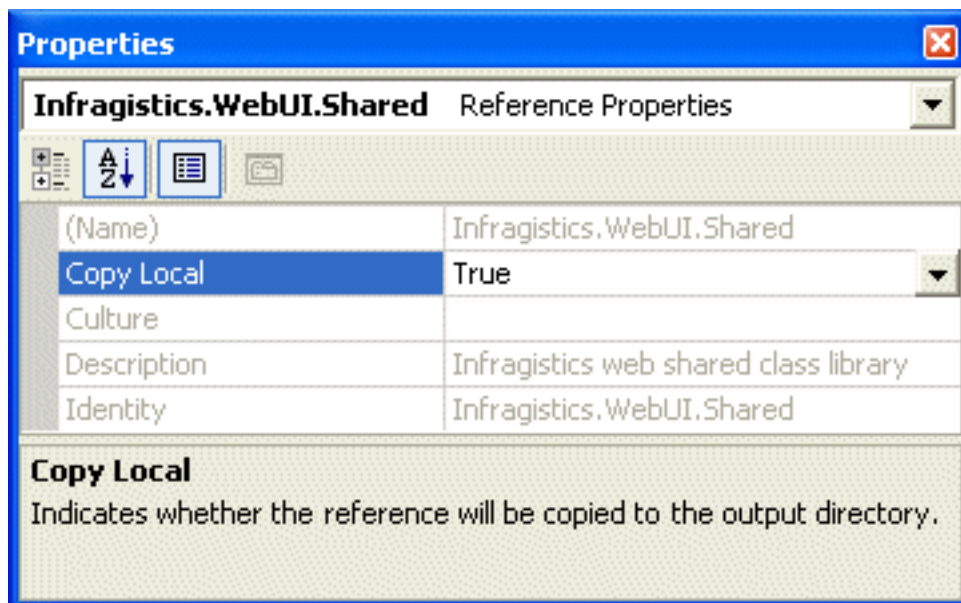
All of the Infragistics controls installed on your development system display on the Infragistics tab. If you have the NetAdvantageSuite installed, your development environment will look something like:



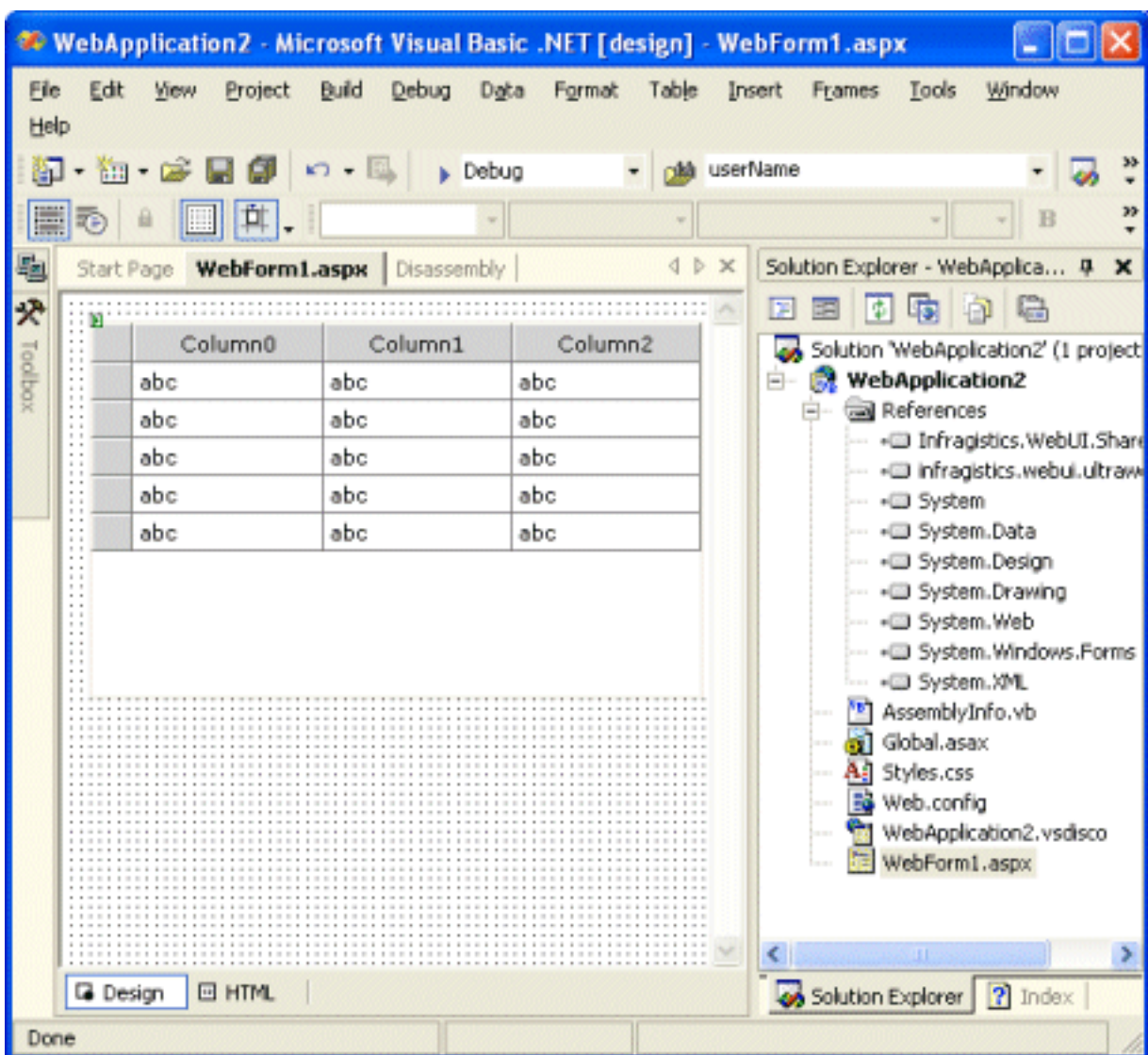
Sample Project

1. Add an instance of UltraWebGrid to WebForm1.aspx
2. Select Design View for WebForm1.aspx
3. Click UltraWebGrid in the Toolbox and drag to the Designer. Notice the grid now displays.
4. Now drag the grid control to the upper left corner of the form.
5. Notice there are two references added to the References folder of the Solutions Explorer. The first is Infragistics.WebUI.Shared and the second is Infragistics.WebUI.UltraWebGrid.v2.
6. Right-click on the Infragistics.WebUI.Shared reference and set the CopyLocal property to True. This will cause the appropriate dll to be copied to the bin directory before the project is launched. This is required since the browser does not look in the GAC (Global Assembly Cache) for the dlls, it only looks in the local bin directory.

Note This step is *VERY IMPORTANT!* If you do not set the **CopyLocal** property correctly, your web application will fail to run.

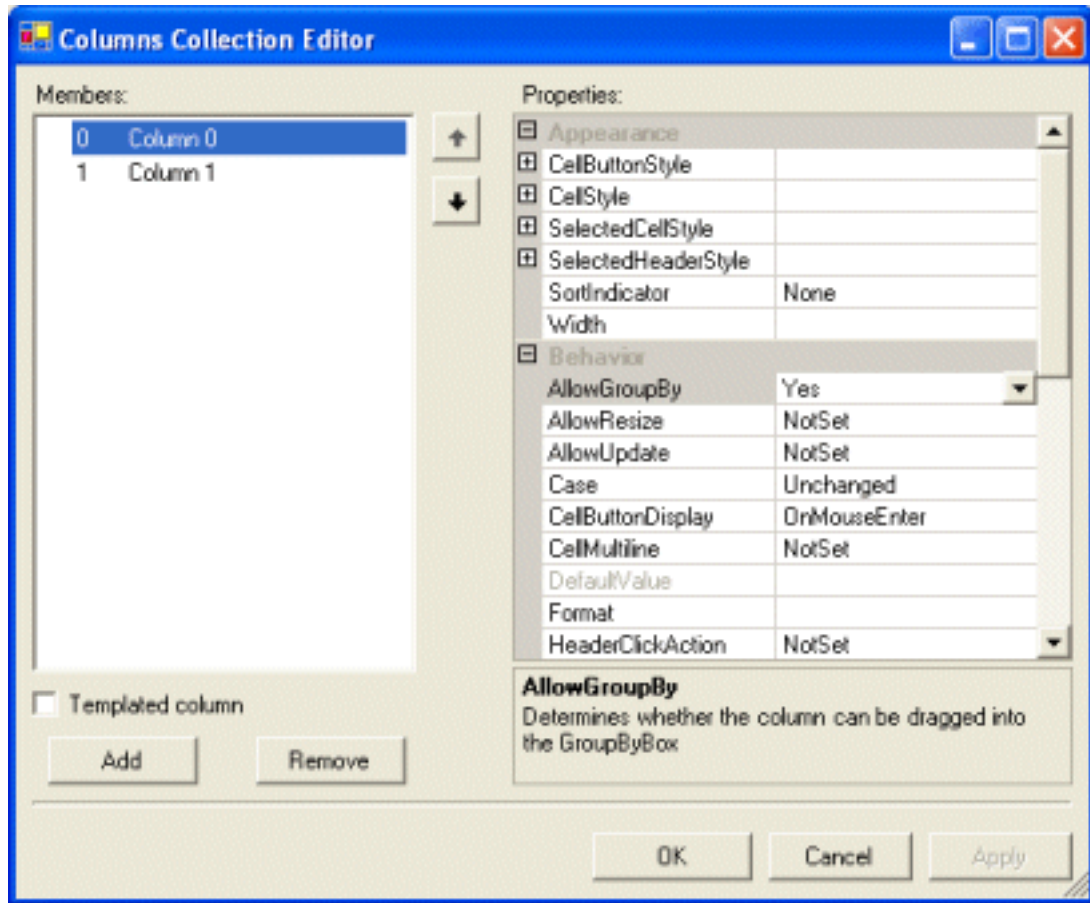


7. You can run the project now, but nothing will display in the browser since the grid does not yet contain any data.
8. Your development environment should now look something like:



Now that you have the control on the form, lets add some columns and rows using the designer:

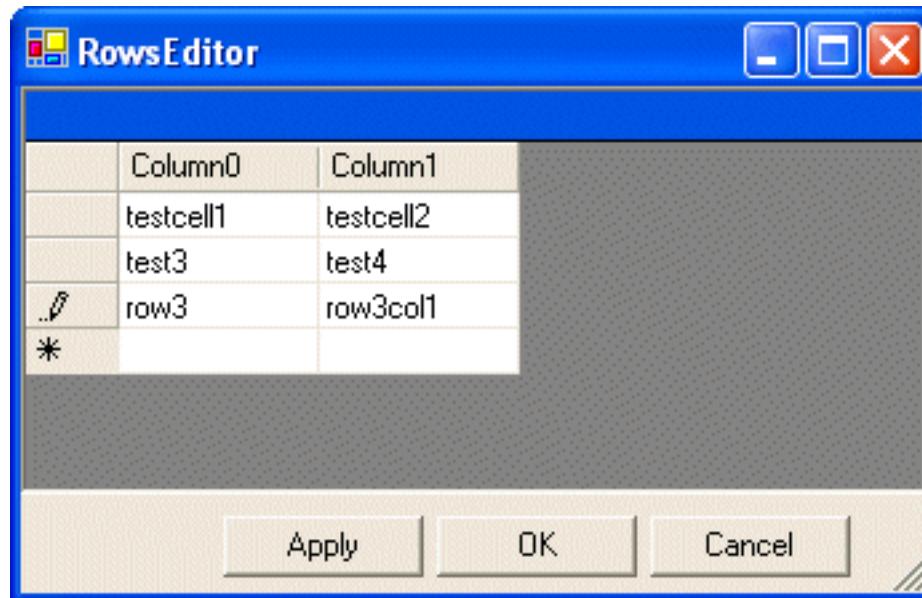
9. Right Click on the control and select "Edit Columns".
10. Click "Add" to add new columns which are given the default names of Column 0, Column 1 etc. and the Columns Collection Editor should look something like:



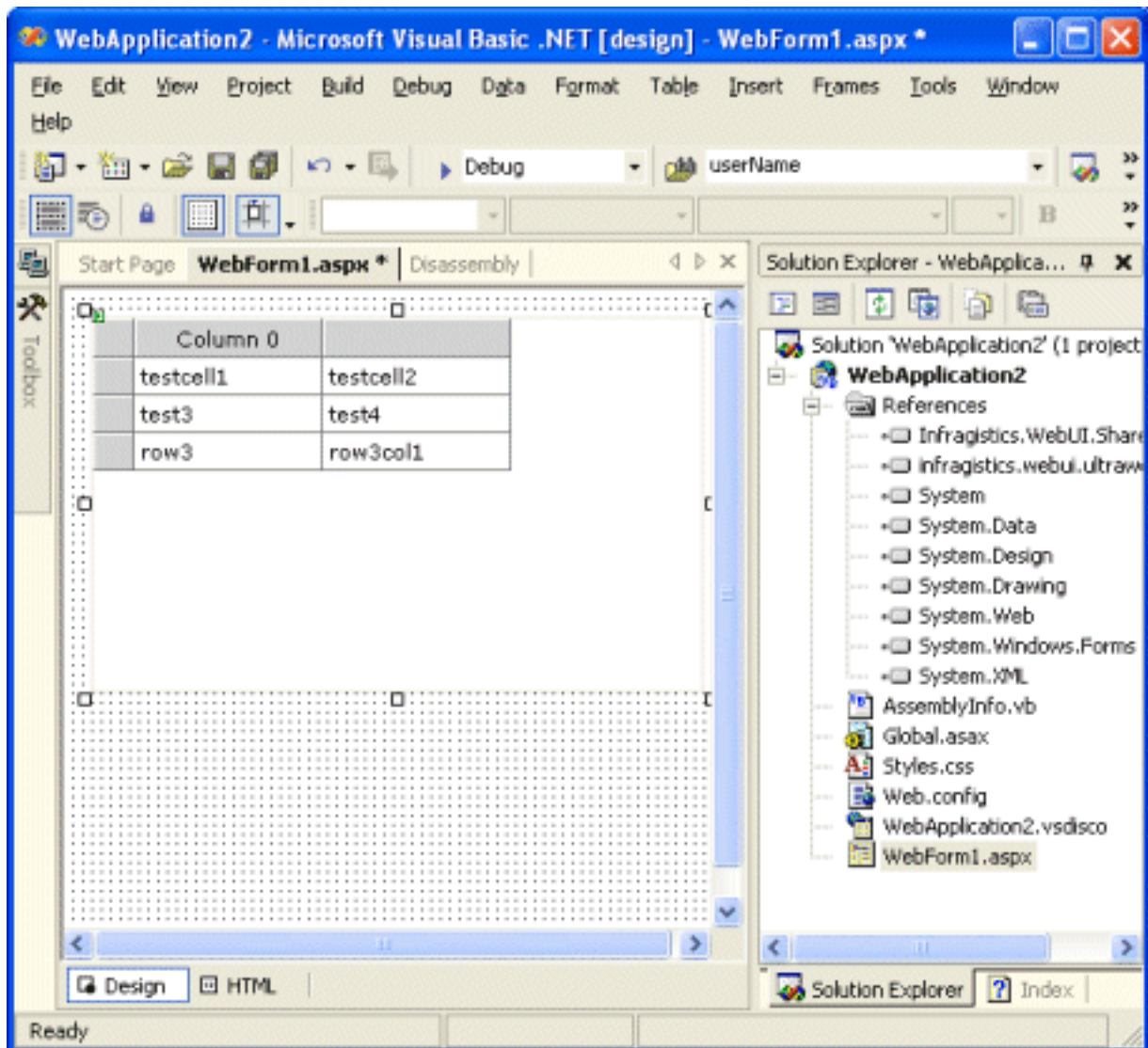
11. Click "OK"

Now you have 2 columns, so lets add some rows and data to the cells:

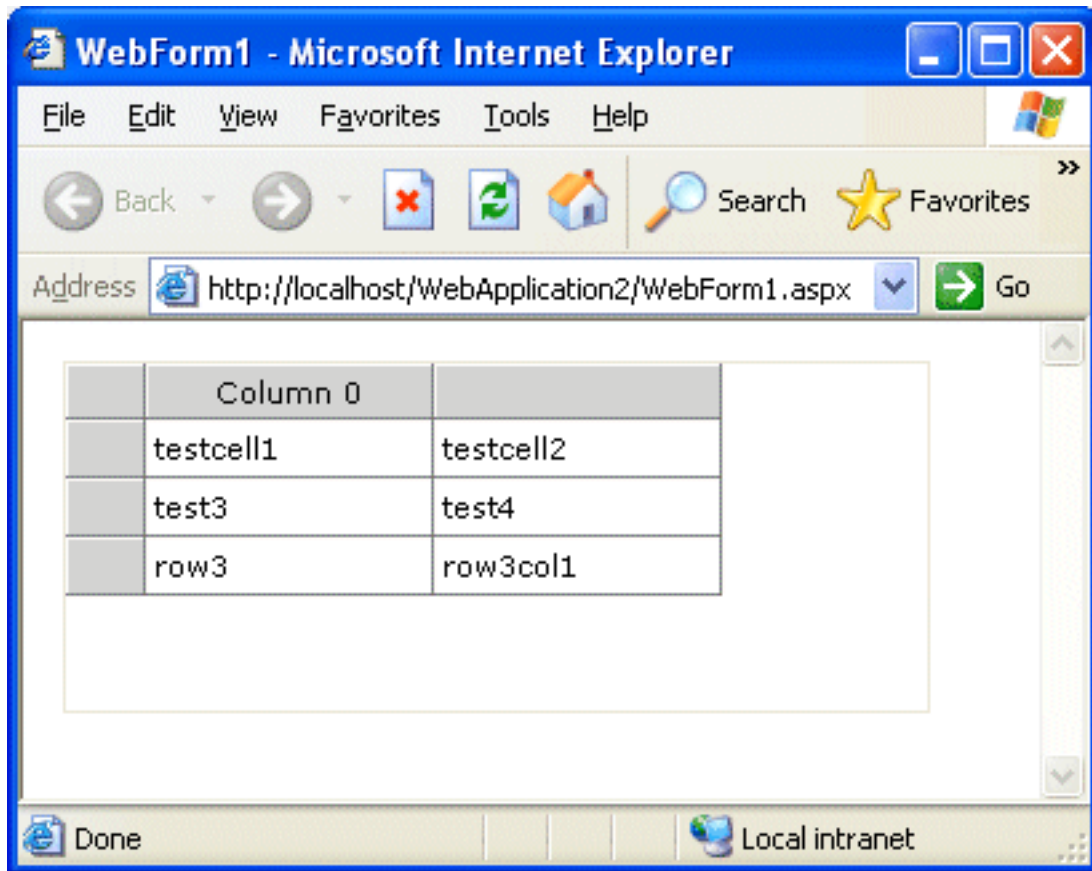
12. Right Click on the Control and Select Edit Rows.
13. At this time you will see the Row Editor and you can enter text into the cells and add rows. Add some text to the cells and add a couple of rows. The Rows Editor should look something like:



14. Click "OK" when you are finished and the development environment should look something like:



15. Run the application and you should see something like:



Code Discussion

There is no code to review in this tutorial.

Review

This tutorial walks the developer through the process of add the UltraWebGrid to a Web Form, adding a new column, and populating some rows and cells using the built-in designer.

Bind To Northwind Flat Data

There are 6 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Code Discussion](#)
- [Review](#)

Background

One common use of web grid controls is the display of data retrieved from a database. This example shows how to retrieve flat (as opposed to hierarchical) data and display it in the grid.

Questions

- How do I retrieve flat data from a data source and bind it to the UltraWebGrid?

Solution

Retrieve a DataTable from the data source. Bind this DataTable to the UltraWebGrid by first setting the **DataSource** property of the grid to the DataTable and then calling the **BindData** method of the grid to place the columns and rows into the HTML output.

The Northwind database is used as a source for the flat data. This tutorial was tested using SQL Server 2000, however any source capable of connecting to a Northwind database should be suitable.

The issue is the creation of a database connection string. After a connection string is created the remainder of the project is quite straightforward.

To allow the developer to connect to various data sources, the project uses OleDb as opposed to SQLClient for data access.

Sample Project

Note This tutorial requires the `uwg2_Helper.vb` module. You can obtain this file by right-clicking on [this link](#) and choosing "Save Target As..." from the context menu. When the File Save dialog appears, save the file in the same folder as the other files that make up the project. You should then add this file to your project.

This project performs the following steps:

1. This sample project loads a persisted database connection string from an XML file. See tutorial [Setting Database Connection String](#) if you have not already created the Northwind database connection string.
2. Creates an OleDb Connection; used to connect to the database.
3. Creates an OleDb DataAdapter; used to retrieve a DataTable containing the data to display in the grid.
4. Binds the DataTable to the grid.

To write this project from scratch, perform the following steps:

1. Start a new Web Forms project.
2. Drag and drop an UltraWebGrid onto the Web Form.
3. Expand the References node of the Solution Explorer and set the Copy Local property of Infragistics.WebUI.Shared to True.

4. Right-click the References node, select "Add Reference" and add the .NET ADODB component reference, and set the **CopyLocal** property to True.
5. Click on the UltraWebGrid on the WebForm and view the Properties pane. Set the **EnableViewState** property to False. This project does not use the grid's ViewState and setting this property to false reduces the amount of data in the rendered page.
6. From the Visual Studio Menu, select Project -> Add Existing Item and add the `uwg_2Helper.vb` module. This module provides classes used to save and retrieve the Northwind database connection string and perform some common tasks.
7. Add the following code to the Page_Load event of the Web Form:

In Visual Basic:

```
' create a data connection
Dim cnNorthwind As New OleDb.OleDbConnection()

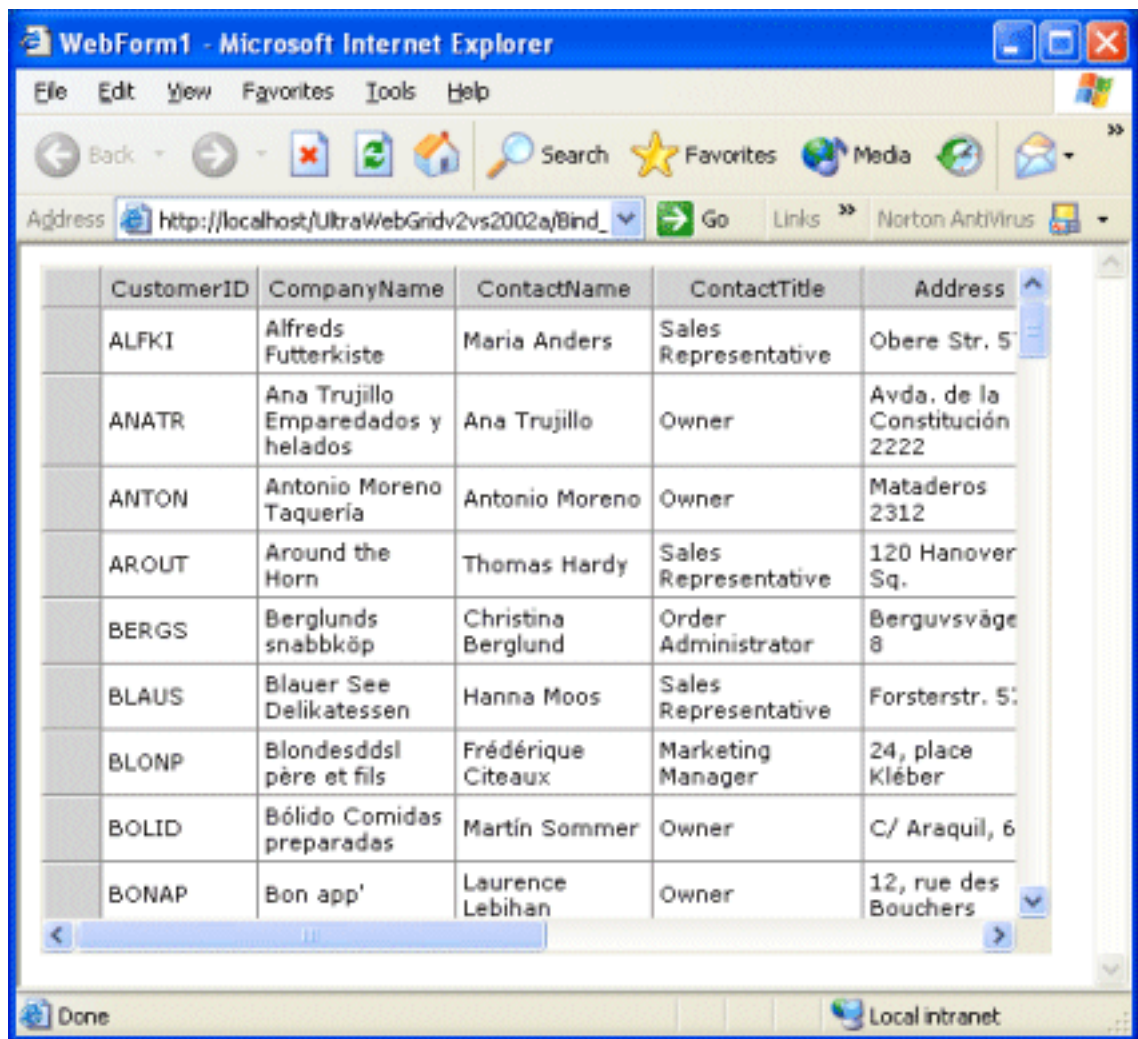
cnNorthwind.ConnectionString = _
UWG2.Utility.DataConnection.ConnectionString()

' create a data adapter
Dim daNorthwindCustomers As OleDb.OleDbDataAdapter = _
UWG2.Utility.OleDbDataAdapter.CreateSelectOnly( _
"SELECT * FROM Customers", cnNorthwind)

' create and populate a data table
Dim dtNorthwindCustomers As New DataTable()
daNorthwindCustomers.Fill(dtNorthwindCustomers)

' bind data table to grid
UltraWebGrid1.DataSource = dtNorthwindCustomers
UltraWebGrid1.DataBind()
```

8. Run the Project and the Web Form should display something like:



Code Discussion

Files containing code in this project:

uwg2_Helper.vb

This code is reviewed in the project *The uwg2Helper.Web Namespace*.

WebForm1.aspx

WebForm1.aspx contains the code relevant to this project and consists of the following code regions:

Form Events

The Form Events Region contains the following event handlers:

- **Page_Load** - The page's **Load** event is used to retrieve the Northwind Customers database and bind it to the grid:
To access the database, the following code creates a new OleDbConnection object and invokes a **ConnectionString** method in the UWG2.Utility namespace which retrieves a persisted connection string.

In Visual Basic:

```
' create a data connection
Dim cnNorthwind As New OleDb.OleDbConnection()
cnNorthwind.ConnectionString = _
UWG2.Utility.DataConnection.ConnectionString()
```


This code declares an OleDbDataAdapter object and invokes a **CreateSelectOnly** method in the UWG2.Utility namespace which returns an initialized Select Only data adapter.

In Visual Basic:

```
' create a data adapter
Dim daNorthwindCustomers As OleDb.OleDbDataAdapter = _
UWG2.Utility.OleDbDataAdapter.CreateSelectOnly( _
"SELECT * FROM Customers", cnNorthwind)
```

To retrieve the data from the database, the following code declares a new DataTable object and invokes the Fill method of the data adapter.

In Visual Basic:

```
' create and populate a data table
Dim dtNorthwindCustomers As New DataTable()
daNorthwindCustomers.Fill(dtNorthwindCustomers)
```

The data table now contains the data you want to bind to the grid. The following code sets the DataSource property of the grid to the data table and invokes the DataBind method of the grid to place the columns and rows into the grid.

In Visual Basic:

```
' bind data table to grid
UltraWebGrid1.DataSource = dtNorthwindCustomers
UltraWebGrid1.DataBind()
```

Review

This walk-through tutorial provides step-by-step instructions on how to retrieve hierarchical data from a data source and bind that data to the UltraWinGrid.

Bind to Northwind Hierarchical Data

There are 6 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Code Discussion](#)
- [Review](#)

Background

One common use of web grid controls is the display of data retrieved from a database. This example shows how to retrieve hierarchical data and display it in the grid.

Questions

- How do I retrieve hierarchical data from a data source and bind it to the UltraWinGrid?

Solution

Retrieve multiple DataTables from the data source. Add the DataTables to a DataSet and declare the relationships between the tables. Bind the DataSet to the UltraWebGrid by first setting the **DataSource** property of the grid to the DataSet and then calling the **BindData** method of the grid to place the columns and rows into the HTML output.

The Northwind database is used as a source for the flat data. This tutorial was tested using SQL Server 2000, however any source capable of connecting to a Northwind database should be suitable.

The issue is the creation of a database connection string. After a connection string is created the remainder of the project is quite straight forward.

To allow the developer to connect to various data sources, the project uses OleDb as opposed to SQLClient for data access.

Sample Project

Note This tutorial requires the `uwg2_Helper.vb` module. You can obtain this file by right-clicking on [this link](#) and choosing "Save Target As..." from the context menu. When the File Save dialog appears, save the file in the same folder as the other files that make up the project. You should then add this file to your project.

This project performs the following steps:

1. This sample project loads a persisted database connection string from an XML file. See tutorial "Setting Database Connection String" if you have not already created the Northwind database connection string.
2. Creates an OleDb Connection; used to connect to the database.
3. Creates an OleDb DataAdapter and retrieves a DataTable containing the Northwind Customers data to display in the grid.

Note Speed issues continue to plague the thin client for many reasons. To help this problem you should be careful to not try to place more than maybe 100 to 250 records in a single form.

4. Creates an OleDb DataAdapter and retrieves a DataTable containing the Northwind Orders data to display in the grid.
5. Binds the Customers and Orders DataTables to a new DataSet.

6. Creates a relationship between the Customers and Orders Data tables and applies it to the DataSet.
7. Binds the DataSet to the grid.

To write this project from scratch, perform the following steps:

1. Start a new Web Forms project.
2. Drag and drop an UltraWebGrid onto the Web Form.
3. Expand the References node of the Solution Explorer and set the **CopyLocal** property of Infragistics.WebUI.Shared to True
4. Right-click the References Node, select "Add Reference" and add the .NET ADODB component reference, and set the **CopyLocal** property to True.
5. Click on the UltraWebGrid on the WebForm and view the Properties pane. Set the **EnableViewState** property to False. This project does not use the grid's ViewState and setting this property to false reduces the amount of data in the rendered page.
6. From the Visual Studio Menu, select ProjectAddExistingItem and add the `uwg2_Helper.vb` module. This module provides classes used to save and retrieve the Northwind database connection string and perform some common tasks. See tutorial The uwg2Helper.Web Namespace
7. Add the following code to the **Page_Load** event of the Web Form:

In Visual Basic:

```
' create a data connection
Dim cnNorthwind As New OleDb.OleDbConnection()
cnNorthwind.ConnectionString = _
UWG2.Utility.DataConnection.ConnectionString()

' create a data adapter for customer data
Dim daNorthwindCustomers As OleDb.OleDbDataAdapter = _
UWG2.Utility.OleDbDataAdapter.Create( _
"SELECT * FROM Customers WHERE CustomerID < 'B'", cnNorthwind)

' create and populate a customer data table
Dim dtNorthwindCustomers As New DataTable("NorthwindCustomers")
daNorthwindCustomers.Fill(dtNorthwindCustomers)

' create a data adapter for order data
Dim daNorthwindOrders As OleDb.OleDbDataAdapter = _
UWG2.Utility.OleDbDataAdapter.Create( _
"SELECT * FROM Orders WHERE CustomerID < 'B'", cnNorthwind)

' create and populate a order data table
Dim dtNorthwindOrders As New DataTable("NorthwindOrders")
daNorthwindOrders.Fill(dtNorthwindOrders)

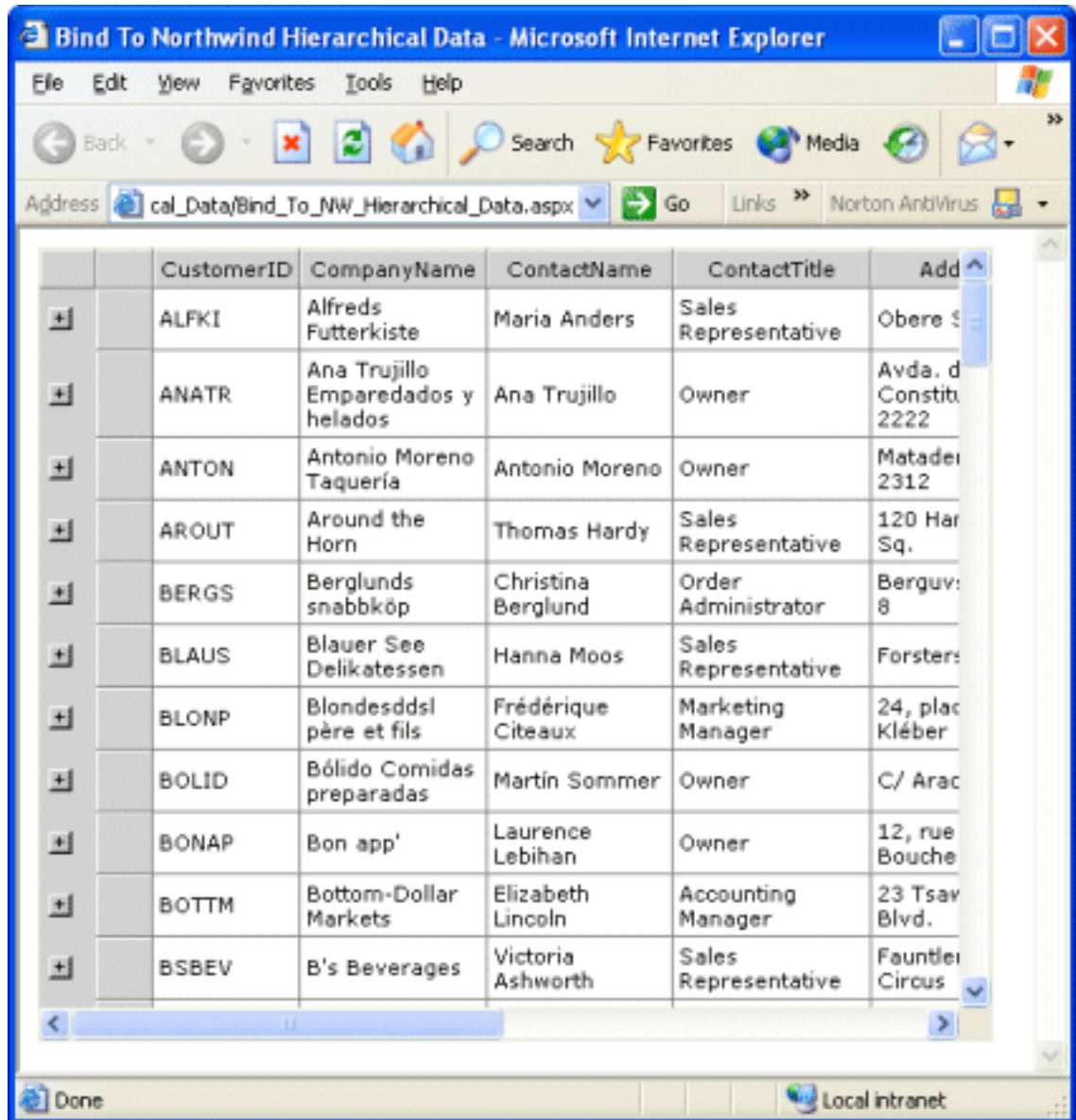
' place both data tables into a dataset
Dim dsNorthwind As New DataSet()
dsNorthwind.Tables.Add(dtNorthwindCustomers)
dsNorthwind.Tables.Add(dtNorthwindOrders)

' create customers/orders relationship and add to DataSet
Dim relCustOrder As New DataRelation("CustOrder" _
, dtNorthwindCustomers.Columns("CustomerID") _
, dtNorthwindOrders.Columns("CustomerID"))
dsNorthwind.Relations.Add(relCustOrder)
```

```
UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.UltraWebGrid.ViewType.Hierarchical
```

```
' bind data table to grid  
UltraWebGrid1.DataSource = dsNorthwind  
UltraWebGrid1.DataBind()
```

8. Run the Project and the Web Form should display something like:



Code Discussion

Files containing code in this project:

uwg2_Helper.vb

This code is reviewed in the project *The uwg2Helper.Web Namespace*.

Bind_To_NW_Hierarchical_Data.aspx

Bind_To_NW_Hierarchical_Data.aspx contains the code relevant to this project and consists of the following code regions:

Form Events

The Form Events Region contains the following event handlers:

- **Page_Load** - The page's **Load** event is used to retrieve the Northwind Customers database and bind it to the grid:
To access the database, the following code creates a new OleDbConnection object and invokes a .ConnectionString method in the UWG2.Utility namespace which retrieves a persisted connection string.

In Visual Basic:

```
' create a data connection
Dim cnNorthwind As New OleDb.OleDbConnection()
cnNorthwind.ConnectionString = _
UWG2.Utility.DataConnection.ConnectionString()
```

This code declares an OleDbDataAdapter object and invokes a **CreateSelectOnly** method in the UWG2.Utility namespace which returns an initialized Select Only data adapter. Note only customerID starting with the character A will display with this sample.

In Visual Basic:

```
' create a data adapter for customer data
Dim daNorthwindCustomers As OleDb.OleDbDataAdapter = _
UWG2.Utility.OleDbDataAdapter.CreateSelectOnly( _
"SELECT * FROM Customers WHERE CustomerID < 'B'", cnNorthwind)
```

To retrieve the customers data from the database; the following code declares a new DataTable object and invokes the Fill method of the data adapter.

In Visual Basic:

```
' create and populate a customer data table
Dim dtNorthwindCustomers As New DataTable("NorthwindCustomers")
daNorthwindCustomers.Fill(dtNorthwindCustomers)
```

This code declares an OleDbDataAdapter object and invokes a **CreateSelectOnly** method in the uwg2Helper namespace which returns an initialized Select Only data adapter.

In Visual Basic:

```
' create a data adapter for order data
Dim daNorthwindOrders As OleDb.OleDbDataAdapter = _
sMatzen.Web.Utility.OleDbDataAdapter.Create( _
"SELECT * FROM Orders WHERE CustomerID < 'B'", cnNorthwind)
```

To retrieve the orders data from the database; the following code declares a new DataTable object and invokes the Fill method of the data adapter.

In Visual Basic:

```
' create and populate a order data table
Dim dtNorthwindOrders As New DataTable("NorthwindOrders")
daNorthwindOrders.Fill(dtNorthwindOrders)
```

To display the two tables as hierarchical data; the following code declares a new DataSet and adds both the Customers and Orders tables.

In Visual Basic:

```
' place both data tables into a dataset
Dim dsNorthwind As New DataSet()
dsNorthwind.Tables.Add(dtNorthwindCustomers)
dsNorthwind.Tables.Add(dtNorthwindOrders)
```

Now that both tables are in the DataSet, a relationship object is created relating the CustomerID of both tables and added to the DataSet.

In Visual Basic:

```
' create customers/orders relationship and add to DataSet
Dim relCustOrder As New DataRelation("CustOrder" _
, dtNorthwindCustomers.Columns("CustomerID") _
, dtNorthwindOrders.Columns("CustomerID"))
dsNorthwind.Relations.Add(relCustOrder)
```

The data set now contains the data you want to bind to the grid. The following code sets the DataSource property of the grid to the data set and invokes the DataBind method of the grid to place the columns and rows into the grid.

In Visual Basic:

```
' bind data set to grid
UltraWebGrid1.DataSource = dsNorthwind
UltraWebGrid1.DataBind()
```

Review

This walk-through tutorial provides step-by-step instructions on how to retrieve hierarchical data from a data source and bind that data to the UltraWinGrid.

Initialize Layout

There are 3 major steps to this exercise:

- [Background](#)
- [Sample Project](#)
- [Review](#)

Background

Many developers use the built-in designers and property pages to configure their grids. Others prefer to perform most of their WebGrid initialization to be performed with code. Placing grid initialization in code provides explicit visualization of what initialization is being performed and allows the developer to remove and replace the control on the designer without losing the initialization settings. However, the advent of being able to make designer settings persistent with XML tends to mitigate this issue (if you remember to save your settings).

There are three primary places where WebGrid initialization settings should be placed:

Before Data Bind

A good rule to follow is if the property is not a part of the DisplayLayout, Bands collection, Columns collection, or Rows collection, set it before **DataBind**. The most important properties that must be set are the **DataSource** and **DataMember** properties.

Initialize Layout Event

InitializeLayout fires when the WebGrid is ready to receive display layout related property settings.

Initialize Row Event

InitializeRow fires when the WebGrid is ready to receive row related property settings and value manipulation. See the tutorial Initialize Row.

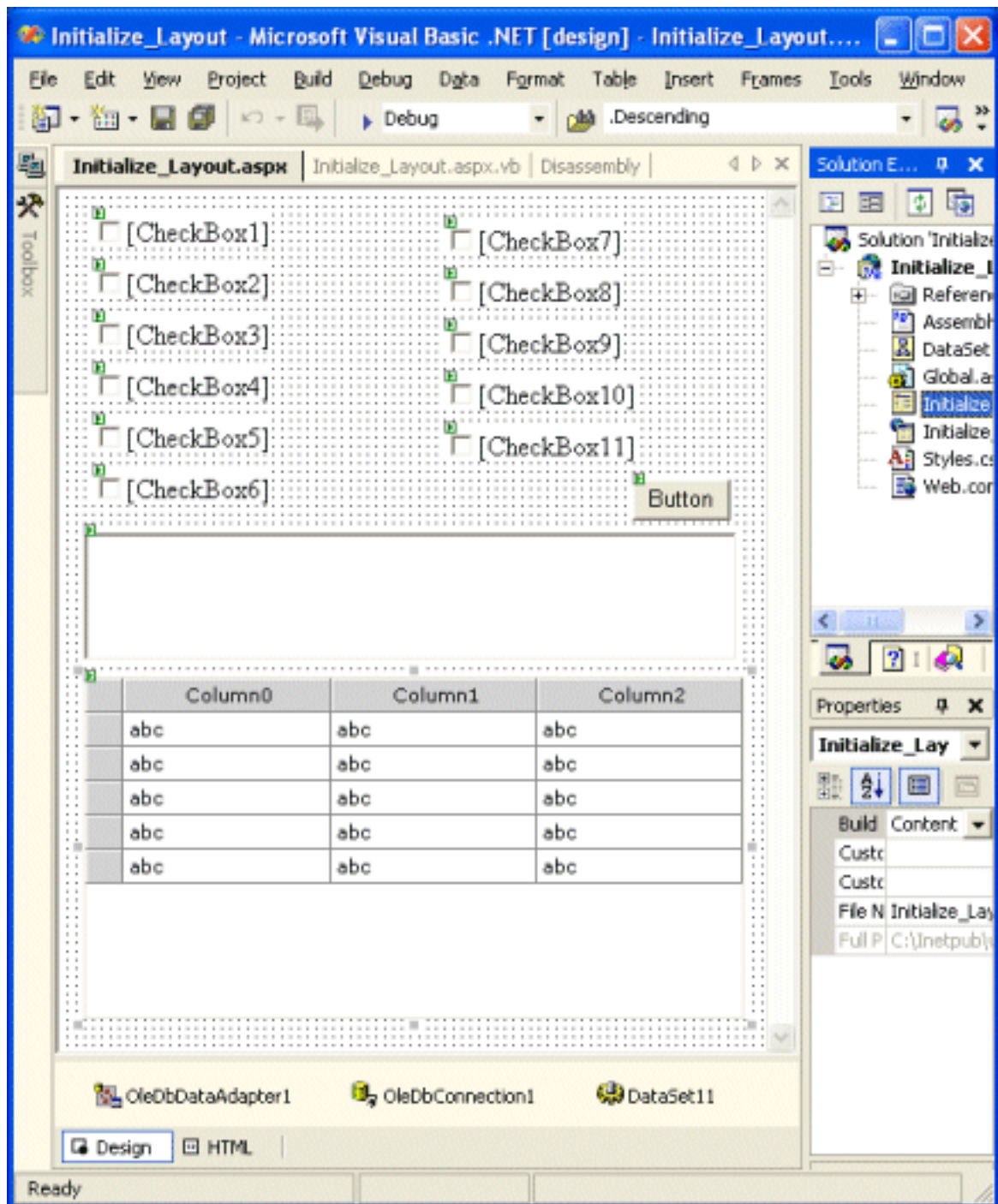
Sample Project

This sample project allows the user to select from a list of Northwind Customers table columns to display. When the Submit button is pressed the WebGrid is populated with these columns. In the **InitializeLayout** event, each column receives some property settings and the columns are moved to a more desirable order.

To create this project, perform the following steps:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb
 - Choose a Query Type: User SQL statements
 - Generate the SQL Statements: Click Query Builder, Add Customers and click Close, click (All Columns) and click OK.
 - Click Finish
3. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.

4. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
5. Click the Infragistics.WebUI.Shared reference and set the following properties:
CopyLocal = True
6. Drag and drop 11 check box controls, a button control, and a text box control onto the designer and arrange the controls something like:



7. Add the following private function. This function creates a custom SQL command to retrieve only the selected columns (and the CustomerID column since it is the primary key and is a constraint on the DataSet). The SQL command is applied to the data adapter select command command text property and the data set is filled from the database:

In Visual Basic:

```
Private Function RetrieveCustomerData() As DataSet
    Dim sbSQL As New System.Text.StringBuilder()

    sbSQL.Append("SELECT")

    If CheckBox1.Checked = True Then sbSQL.Append(" CompanyName,")
    If CheckBox2.Checked = True Then sbSQL.Append(" ContactName,")
    If CheckBox3.Checked = True Then sbSQL.Append(" ContactTitle,")
    If CheckBox4.Checked = True Then sbSQL.Append(" Phone,")
    If CheckBox5.Checked = True Then sbSQL.Append(" Fax,")
    If CheckBox6.Checked = True Then sbSQL.Append(" Address,")
    If CheckBox7.Checked = True Then sbSQL.Append(" City,")
    If CheckBox8.Checked = True Then sbSQL.Append(" Region,")
    If CheckBox9.Checked = True Then sbSQL.Append(" PostalCode,")
    If CheckBox10.Checked = True Then sbSQL.Append(" Country,")

    sbSQL.Append(" CustomerID") ' required due to primary key constraint

    ' append remainder of sql statement
    sbSQL.Append(" FROM Customers")

    ' place SQL statement into text box
    TextBox1.Text = sbSQL.ToString

    ' apply command text and retrieve dataset
    OleDbDataAdapter1.SelectCommand.CommandText = sbSQL.ToString
    OleDbDataAdapter1.Fill(DataSet11)

    Return DataSet11
End Function
```

8. Add the following code to the page load event. On first page load the check boxes, button and text box properties are set, and on all page loads the customer data is retrieved, bound to the WebGrid data source and the data bind method invoked:

In Visual Basic:

```
If Me.IsPostBack = False Then

    ' set properties of check boxes
    CheckBox1.Text = "Company Name"
    CheckBox1.Checked = True
    CheckBox2.Text = "Contact Name"
    CheckBox2.Checked = True
    CheckBox3.Text = "Contact Title"
    CheckBox4.Text = "Phone"
    CheckBox4.Checked = True
    CheckBox5.Text = "Fax"
    CheckBox6.Text = "Address"
    CheckBox7.Text = "City"
    CheckBox8.Text = "Region"
    CheckBox9.Text = "Postal Code"
    CheckBox10.Text = "Country"
    CheckBox11.Text = "CusotmerID"

    ' set text for submit button
    Button1.Text = "Submit"
```

```
' set text box properties
TextBox1.TextMode = TextBoxMode.MultiLine
```

End If

```
' retrieve data from database
UltraWebGrid1.DataSource = RetrieveCustomerData()
```

```
' bind data to web grid
UltraWebGrid1.DataBind()
```

9. Add the following code to the WebGrid **InitializeLayout** event. This code sets some properties of each column:

In Visual Basic:

With e.Layout

```
' set properties of each individual column
With .Bands(0).Columns.FromKey("CompanyName")
    .Move(0)
    If CheckBox1.Checked = False Then
        .Hidden = True
    Else
        .Hidden = False
    End If
End With
```

```
With .Bands(0).Columns.FromKey("ContactName")
    .Move(1)
    If CheckBox2.Checked = False Then
        .Hidden = True
    Else
        .Hidden = False
    End If
End With
```

```
With .Bands(0).Columns.FromKey("ContactTitle")
    .Move(2)
    If CheckBox3.Checked = False Then
        .Hidden = True
    Else
        .Hidden = False
    End If
End With
```

```
With .Bands(0).Columns.FromKey("Phone")
    .Move(3)
    If CheckBox4.Checked = False Then
        .Hidden = True
    Else
        .Hidden = False
    End If
End With
```

```
With .Bands(0).Columns.FromKey("Fax")
    .Move(4)
    If CheckBox5.Checked = False Then
        .Hidden = True
    Else
        .Hidden = False
    End If
End With
```

```

        End If
    End With

    With .Bands(0).Columns.FromKey("Address")
        .Move(5)
        If CheckBox6.Checked = False Then
            .Hidden = True
        Else
            .Hidden = False
        End If
    End With

    With .Bands(0).Columns.FromKey("City")
        .Move(6)
        If CheckBox7.Checked = False Then
            .Hidden = True
        Else
            .Hidden = False
        End If
    End With

    With .Bands(0).Columns.FromKey("Region")
        .Move(7)
        If CheckBox8.Checked = False Then
            .Hidden = True
        Else
            .Hidden = False
        End If
    End With

    With .Bands(0).Columns.FromKey("PostalCode")
        .Move(8)
        If CheckBox9.Checked = False Then
            .Hidden = True
        Else
            .Hidden = False
        End If
    End With

    With .Bands(0).Columns.FromKey("Country")
        .Move(9)
        If CheckBox10.Checked = False Then
            .Hidden = True
        Else
            .Hidden = False
        End If
    End With

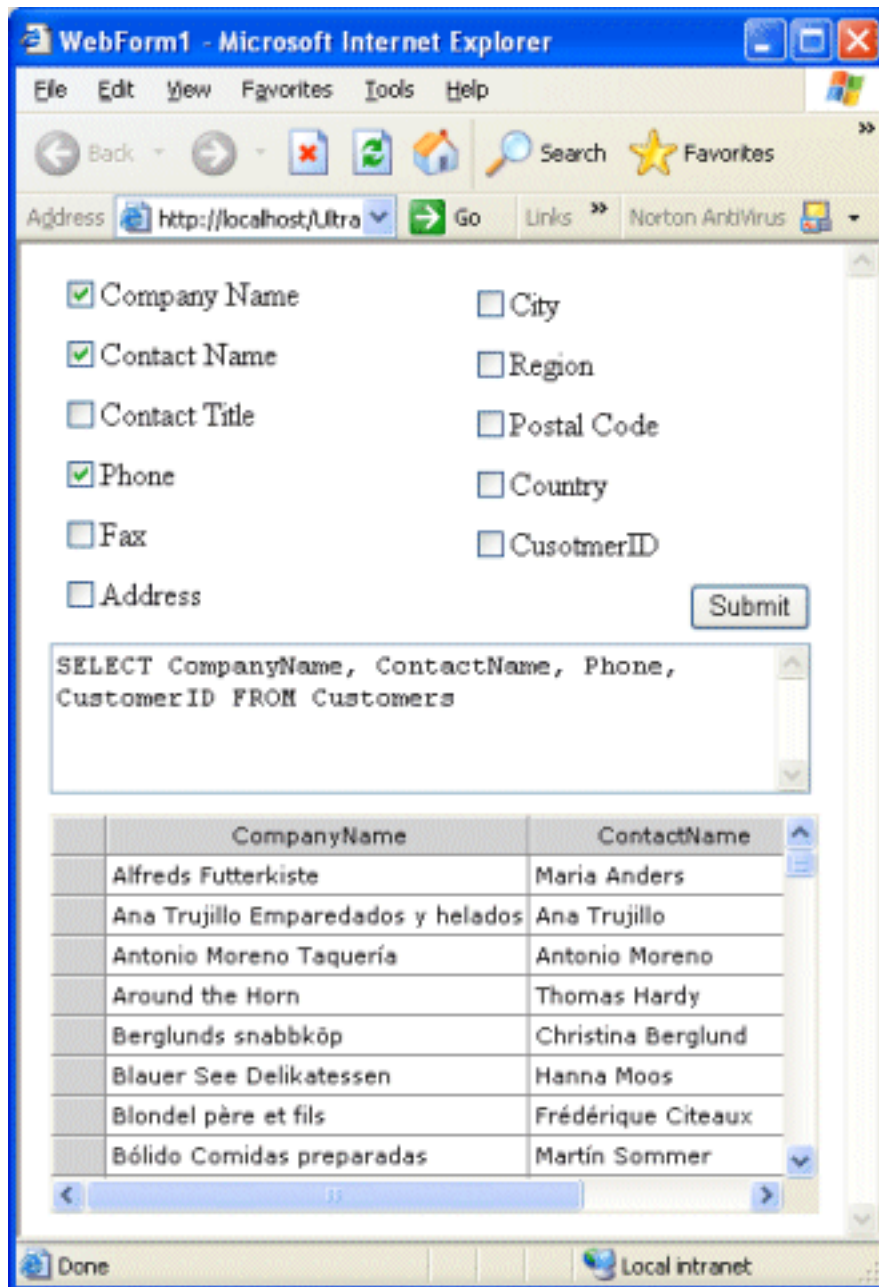
    With .Bands(0).Columns.FromKey("CustomerID")
        .Move(10)
        If CheckBox11.Checked = False Then
            .Hidden = True
        Else
            .Hidden = False
        End If
    End With
End With

```

Note You will notice this code is voluminous with 8 lines of code for each column. This can be reduced significantly with a loop through the columns collection, however when you loop through the columns collection you may run into trouble with the Move method of the column object. Rather than setting a

visible position in the column object, the **Move** method moves the column in the collection, thus disrupting the sequence within the collection you are looping through. This will cause you to process some columns twice and not process other columns. So, as long as you are not moving the columns the loop approach should work.

10. Run the project and you should see something like:



11. Change the column selections and click Submit and observe the new column selections display in the WebGrid.

Review

This tutorial describes the use of the **InitializeLayout** event for setting grid properties and prioritizes the location of various code based property settings. The moving of columns from one position to another is also reviewed.

Initialize Row

There are 6 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

There are times when the underlying data does not contain all of the columns of information required to populate the grid. In many of these applications the best choice is to add an unbound column to the grid and populate the row values in the **InitializeRow** event.

The **InitializeRow** event is also a good place to set appearance properties of the row that are based on values within the row. For instance, the user may want all rows with a discount of greater than 10% to be highlighted with a different background color.

Questions

- How do I create a new column in the WebGrid and populate the values of this column based on a combination of values in the row?

Solution

Add a new column in the InitializeLayout event and populate the column values in the InitializeRow event.

Sample Project

This sample project uses the Northwind OrderDetails data table to populate a WebGrid with an unbound column and row specific background color highlights.

To create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - o On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb
 - o Choose a Query Type: User SQL statements.
 - o Generate the SQL Statements: Click Query Builder, Add OrderDetails and click Close, click (All Columns) and click OK.
 - o Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.

6. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

7. Click on the WebGrid and set the following properties:

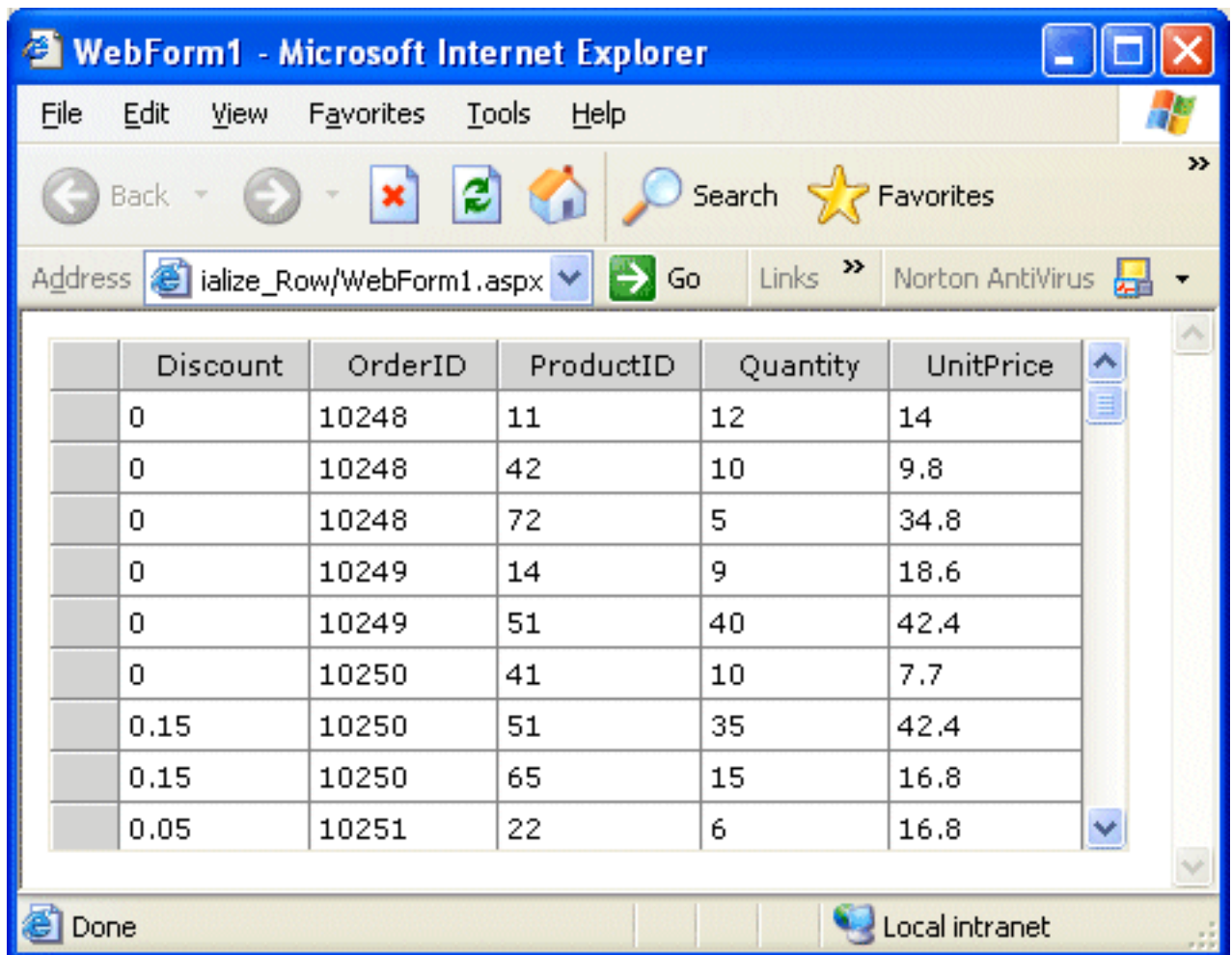
DataSource = DataSet11 **DataMember** = OrderDetails

8. Add the following code to the page's **Load** event to set the command text for this query to load only the top 100 rows, invoke the data adapter fill method to load the dataset and bind the dataset to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbSelectCommand1.CommandText _
    = "SELECT TOP 100 * FROM [Order Details]"
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

9. Run the project and depending on which version of the Northwind database you connect to you should see something like:



The screenshot shows a Microsoft Internet Explorer browser window titled "WebForm1 - Microsoft Internet Explorer". The address bar shows the URL "ialize_Row/WebForm1.aspx". The main content area displays a table with the following data:

	Discount	OrderID	ProductID	Quantity	UnitPrice
	0	10248	11	12	14
	0	10248	42	10	9.8
	0	10248	72	5	34.8
	0	10249	14	9	18.6
	0	10249	51	40	42.4
	0	10250	41	10	7.7
	0.15	10250	51	35	42.4
	0.15	10250	65	15	16.8
	0.05	10251	22	6	16.8

This default data is adequate, but the user most likely wants to see is the OrderID, ProductID, Quantity, UnitPrice, Discount and Net. The Net is calculated by multiplying the Quantity by the UnitPrice by the Discount and rounding to two decimal positions. Since there is no Net column, you must add an unbound column. You can clean up some of the formatting at the same time.

There is more than one way get the Net calculated and into the grid. One approach is to add a column

to the underlying data, pass each row of the underlying data and calculate the column. This tutorial uses the WebGrid **InitializeRow** event to perform the calculation of the Net, and set the background color of all rows with over 10% discount to pink.

10. Add the following code to the **InitializeLayout** event. This code sets some properties of each column and repositions them to the users specifications. It also adds an unbound column to contain the Net price:

In Visual Basic:

With e.Layout.Bands(0).Columns

```
.FromKey("OrderID").HeaderText = "Order"
.FromKey("OrderID").CellStyle.HorizontalAlign = HorizontalAlign.Right
.FromKey("OrderID").Move(0)

.FromKey("ProductID").HeaderText = "Prod"
.FromKey("ProductID").CellStyle.HorizontalAlign = HorizontalAlign.Right
.FromKey("ProductID").Move(1)

.FromKey("Quantity").HeaderText = "Qty"
.FromKey("Quantity").CellStyle.HorizontalAlign = HorizontalAlign.Right
.FromKey("Quantity").Move(2)

.FromKey("UnitPrice").HeaderText = "Price"
.FromKey("UnitPrice").CellStyle.HorizontalAlign = HorizontalAlign.Right
.FromKey("UnitPrice").Format = "c"
.FromKey("UnitPrice").Move(3)

.FromKey("Discount").HeaderText = "Disc"
.FromKey("Discount").CellStyle.HorizontalAlign = HorizontalAlign.Right
.FromKey("Discount").Format = "p"
.FromKey("Discount").Move(4)

' add an unbound column
.Add("Net", "Net")
.FromKey("Net").CellStyle.HorizontalAlign = HorizontalAlign.Right
.FromKey("Net").Format = "c"
```

End With

11. Add the following code to the **InitializeRow** event. This code calculates the value of the Net column from the Quantity, UnitPrice and Discount columns, then if the Discount is greater than 10% the back color of the row is changed to pink:

In Visual Basic:

With e.Row.Cells

```
.FromKey("Net").Value _
= Math.Round(CDec(.FromKey("Quantity").Value) _
* CDec(.FromKey("UnitPrice").Value) _
* (1.0 - CDec(.FromKey("Discount").Value)), 2)

If CDec(.FromKey("Discount").Value) > 0.1 Then
    .FromKey("Discount").Row.Style.BackColor = Color.Pink
End If
End With
```

12. Run the project and you should see something like:

WebForm1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites

Address [lize_Row/WebForm1.aspx](#) Go Links Norton AntiVirus

Order	Prod	Qty	Price	Disc	Net
10248	11	12	\$14.00	0.00 %	\$168.00
10248	42	10	\$9.80	0.00 %	\$98.00
10248	72	5	\$34.80	0.00 %	\$174.00
10249	14	9	\$18.60	0.00 %	\$167.40
10249	51	40	\$42.40	0.00 %	\$1,696.00
10250	41	10	\$7.70	0.00 %	\$77.00
10250	51	35	\$42.40	15.00 %	\$1,261.40
10250	65	15	\$16.80	15.00 %	\$214.20
10251	22	6	\$16.80	5.00 %	\$95.76

Done Local intranet

Review

This tutorial shows how to add an unbound column to the WebGrid in the **InitializeLayout** event and populate that column with values in the **InitializeRow** event. This methodology has many uses when any column in a row needs to be based on row values and is not a part of the underlying data.

Band Appearance

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

When the WebGrid is displaying hierarchical (multi-band) data, the developer may want to set the appearance styles within each band differently. This capability is supported both through the designers and through code by setting the properties of each band differently. With the designer, the developer can open the Bands Collection Editor to set the properties of each band.

Questions

- How can I set the appearance style properties of each band differently?

Solution

Use code in the **InitializeLayout** event to change the appearance properties of each band as the WebGrid is being bound to data.

Use the Band Collection Editor to set properties of the different bands at design-time.

Sample Project

This sample project uses the Northwind Customers and Orders data tables as a source of hierarchical data.

To create this project from scratch, perform the following steps:

1. Start a new Web Forms project.
2. From ToolboxInfragistics, drag a WebGrid to the WebForm1.aspx designer. Move the grid to the upper left corner of the designer and size as desired. Set the following WebGrid properties:

DisplayLayout.**ViewType** = Hierarchical

3. Open the References node in the Solutions Explorer and notice there are two Infragistics entries. Select Infragistics.WebUI.Shared entry and set the following properties:

CopyLocal = True

4. Add a data adapter, data connection controls for the Northwind Customers data. To restrict the number of records being loaded into the grid, add the following where clause to the end of the **CommandText** Property:

```
WHERE (CustomerID < C)
```

For more details on connecting to data using designers see the [Connect to Data Using Designers](#) tutorial.

5. Add a data adapter control for the Northwind Orders data. To restrict the number of records being loaded into the grid, add the following where clause to the end of the **CommandText** Property:

```
WHERE (CustomerID < C)
```

6. Add a typed dataset containing the Customers and Orders data tables and a relationship between the tables on the CustomerID.

7. Add a dataset control to the WebForm1.aspx designer.
8. Select Build -> Rebuild Solution from the main menu to synchronize all aspects of the Visual Studio environment.
9. Select the WinGrid on the WebForm1.aspx designer and set the following properties:
 - DataSource** = DataSet11
 - DataMember** = Customers
10. Add the following code to the **Load** event of the Page:

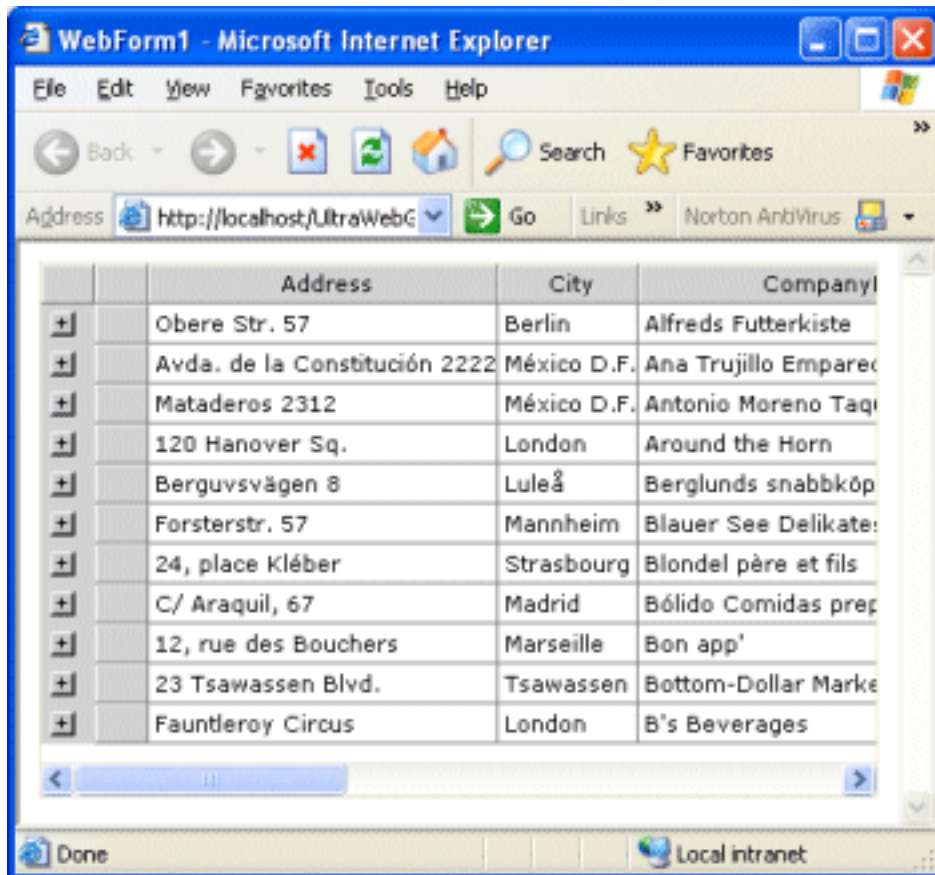
In Visual Basic:

```

If Me.IsPostBack = False Then
    Me.OleDbDataAdapter1.Fill(DataSet11, "Customers")
    Me.OleDbDataAdapter2.Fill(DataSet11, "Orders")
    Me.UltraWebGrid1.DataBind()
End If

```

11. Run the Project and the Web Form should display something like:



Stop the project when you are done.

12. Click on the WebGrid in the WebForm1.aspx designer and open the Bands Collection Editor by clicking on the Bands property collection, then clicking on the ellipse button and Visual Studio should display something like:

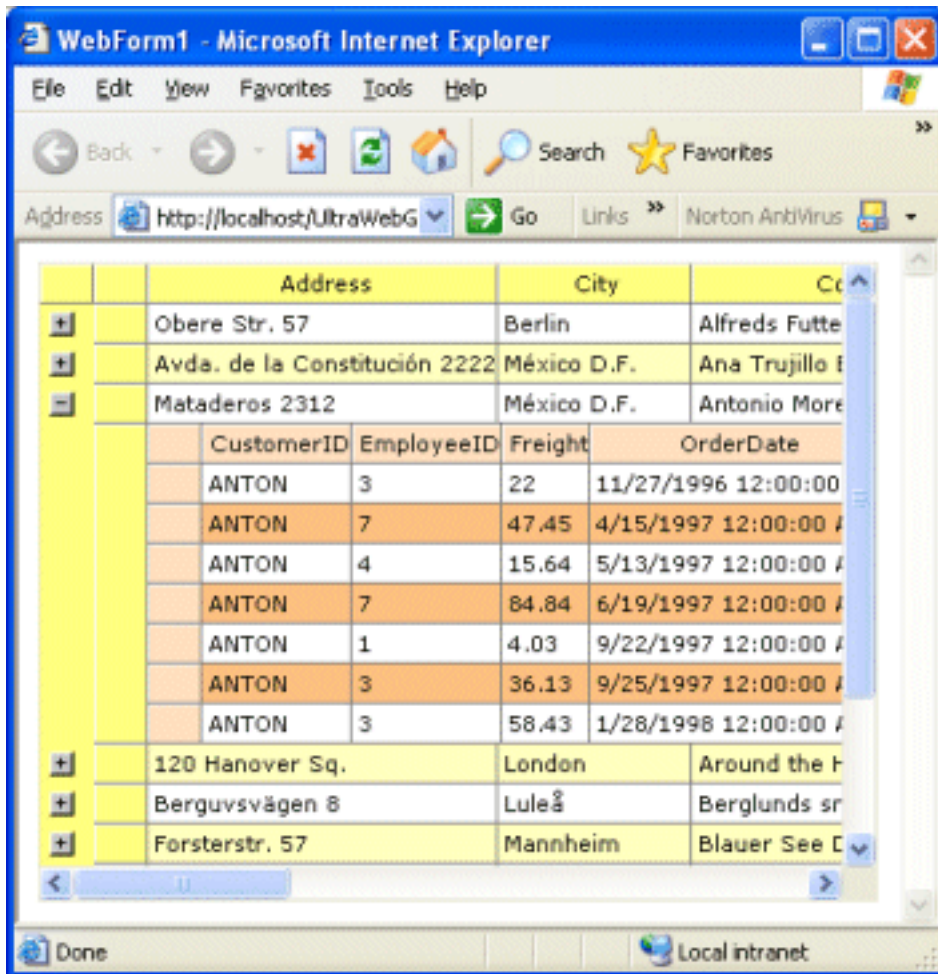


13. From this editor you have access to all of the properties of each band. For both the Customers and Orders bands, set the back color of the RowAlternateStyle and HeaderStyle to something like:

Customers.**HeaderStyle.BackColor** = (dark yellow)
Customers.**RowAlternateStyle.BackColor** = (light yellow)

Orders.**HeaderStyle.BackColor** = (dark pink)
Orders.**RowAlternateStyle.BackColor** = (light pink)

14. Click OK to close the Bands Collection Editor. Run the project and expand one of the Customer nodes. Observe the colors applied to the bands:



Stop the project.

15. Band appearance properties can also be set in code. Set some font properties of both bands with code by adding the following to the WebGrid **InitializeLayout** event:

In Visual Basic:

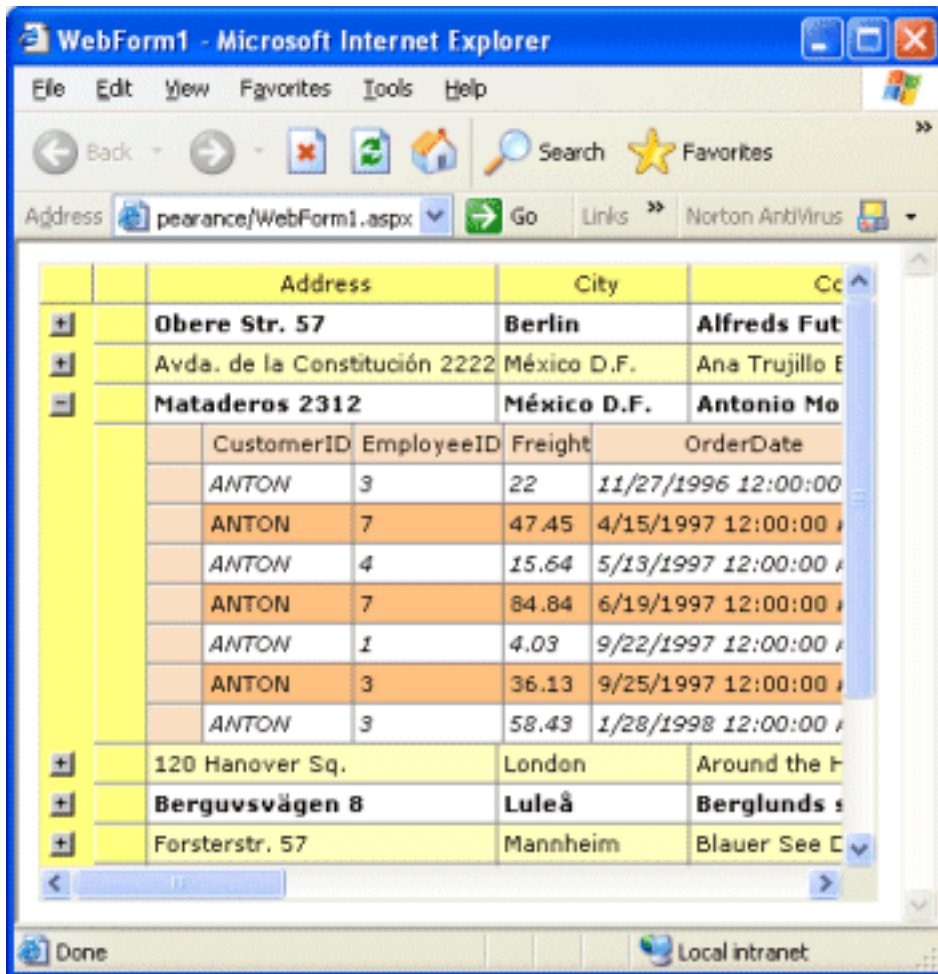
```

' set properties of the Customers band
With e.Layout.Bands.FromKey("Customers")
    .RowStyle.Font.Bold = True
End With

' set properties of the Orders band
With e.Layout.Bands.FromKey("Orders")
    .RowStyle.Font.Italic = True
End With

```

16. Run the project and observe the font appearance changes made with code:



Stop the project.

Review

This tutorial shows how to modify band appearance with both the property editors and with code.

Row Appearance

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Many users like to see every other row with a different background color, and the WebGrid elegantly supports this with the `DisplayLayout.RowAlternateStyleDefault` property. However, there are times when the user may want to see special rows highlighted differently.

Questions

- How do I set the appearance style of rows?
- How do I set the background color of every other row?
- How do I set the appearance style of rows based on cell values?

Solution

- Set the individual properties of the `DisplayLayout.RowStyleDefault` property to change the row appearance.
- Set the individual properties of the `DisplayLayout.RowAlternateStyleDefault` property to change the appearance of every other row.
- In the `InitializeRow` event, set the individual properties of the `e.Layout.Style` property to change the appearance of a row based on cell values.

Sample Project

This sample project uses the Northwind Orders data table as a source of data to illustrate the use of row appearance.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an `OleDbDataAdapter` to the `WebForm1.aspx` designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the `NWind.mdb` file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK From the Toolbox, drag an

UltraWebGrid to the WebForm1.aspx designer, position and size as desired.

4. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.

5. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

6. Click on the WebGrid and set the following properties:

DataSource = DataSet11

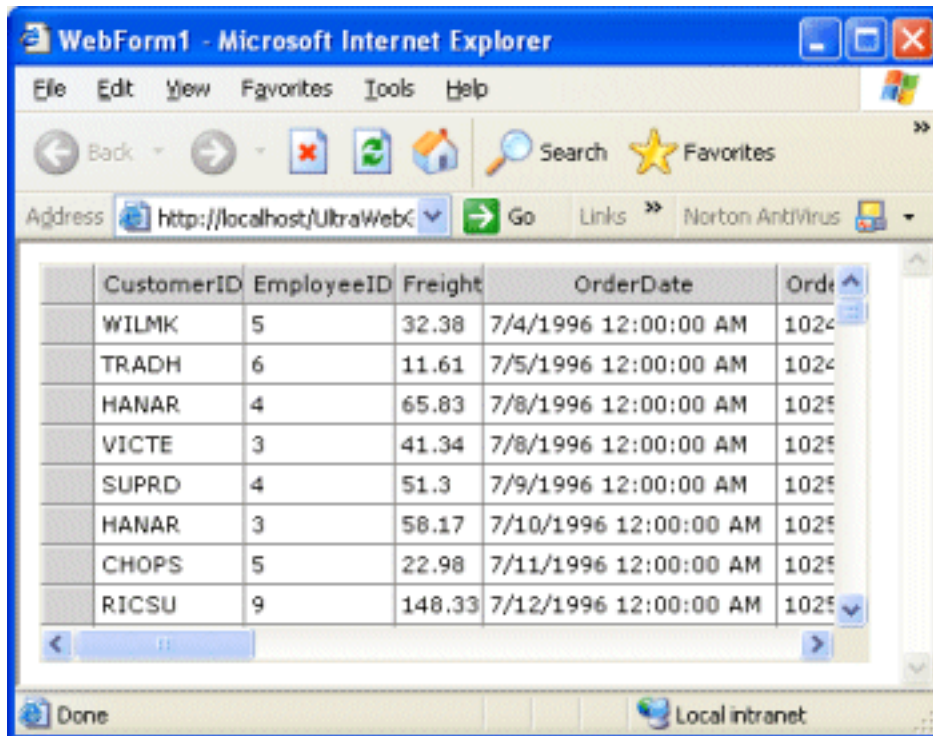
DataMember = Orders

7. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

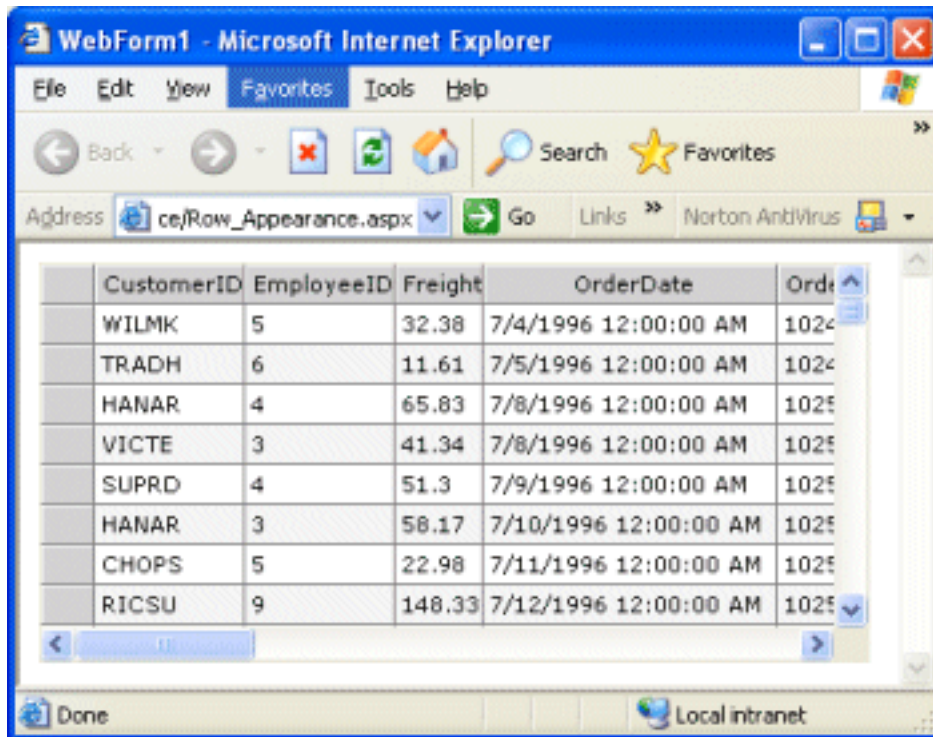
```
If Me.IsPostBack = False Then
    Me.OleDbSelectCommand1.CommandText _
    = "SELECT TOP 100 * FROM [Orders]"
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

8. Run the project and depending on which version of the Northwind database you connect to you should see something like:



CustomerID	EmployeeID	Freight	OrderDate	OrderID
WILMK	5	32.38	7/4/1996 12:00:00 AM	1024
TRADH	6	11.61	7/5/1996 12:00:00 AM	1024
HANAR	4	65.83	7/8/1996 12:00:00 AM	1025
VICTE	3	41.34	7/8/1996 12:00:00 AM	1025
SUPRD	4	51.3	7/9/1996 12:00:00 AM	1025
HANAR	3	58.17	7/10/1996 12:00:00 AM	1025
CHOPS	5	22.98	7/11/1996 12:00:00 AM	1025
RICSU	9	148.33	7/12/1996 12:00:00 AM	1025

9. The developer can set the appearance of all rows through the designer by setting properties in the DisplayLayout.**RowStyleDefault** property.
10. Set the row alternate appearance so that every other row has a different background color. Click the WebGrid on the WebForm1.aspx designer. On the Properties dialog set the following property:
DisplayLayout.RowAlternateStyleDefault = "WhiteSmoke"
11. Run the project and observe the different appearance of every other row:



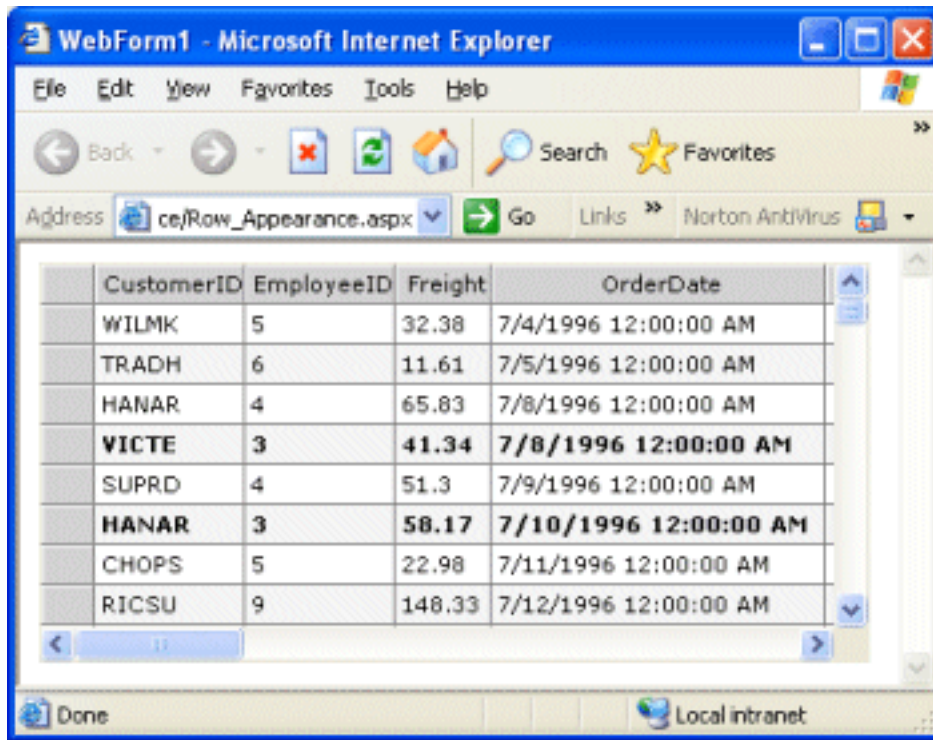
Stop the project.

12. To illustrate the use of appearance properties on each individual row, tell the grid to display the font in bold for all rows with EmployeeID value of 3. This can be accomplished by adding the following code to the WebGrid InitializeRow event:

In Visual Basic:

```
With e.Row.Style
    If e.Row.Cells.FromKey("EmployeeID").Value = 3 Then
        .Font.Bold = True
    End If
End With
```

13. Run the project and observe the bold font in all rows where the EmployeeID has a value of 3:



Review

This project shows how to set the row alternate appearance and selectively change the appearance of a row based on a cell value.

Row Alternate Appearance

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Readability of a grid can be enhanced by having the appearance of every-other row altered slightly. Most commonly the back color of every-other row is different.

Questions

- How can I set the back color of every-other row to a different color?

Solution

Using the WebGrid property page, set the DisplayLayout. RowAlternateStyleDefault.BackColor property to whatever color you would like.

Sample Project

This sample project displays the Northwind Customers data table with the background color of every-other row slightly different.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements
 - Generate the SQL Statements: Click Query Builder, Add Customers and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
6. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

7. Click on the WebGrid and set the following properties:

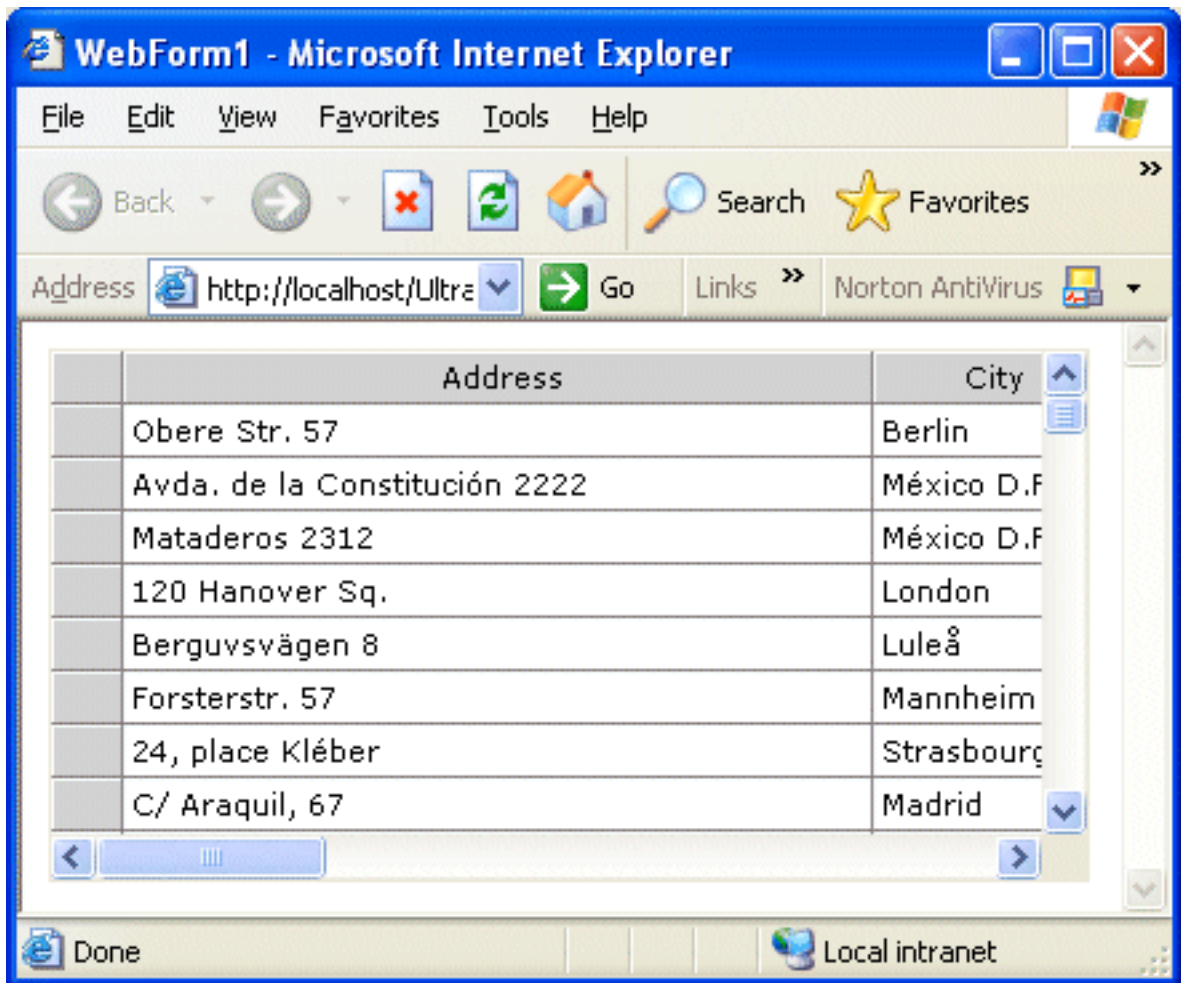
DataSource = DataSet11
DataMember = Customers

8. Add the following code to the page's **Load** event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    OleDbDataAdapter1.Fill(DataSet11)
    UltraWebGrid1.DataBind()
End If
```

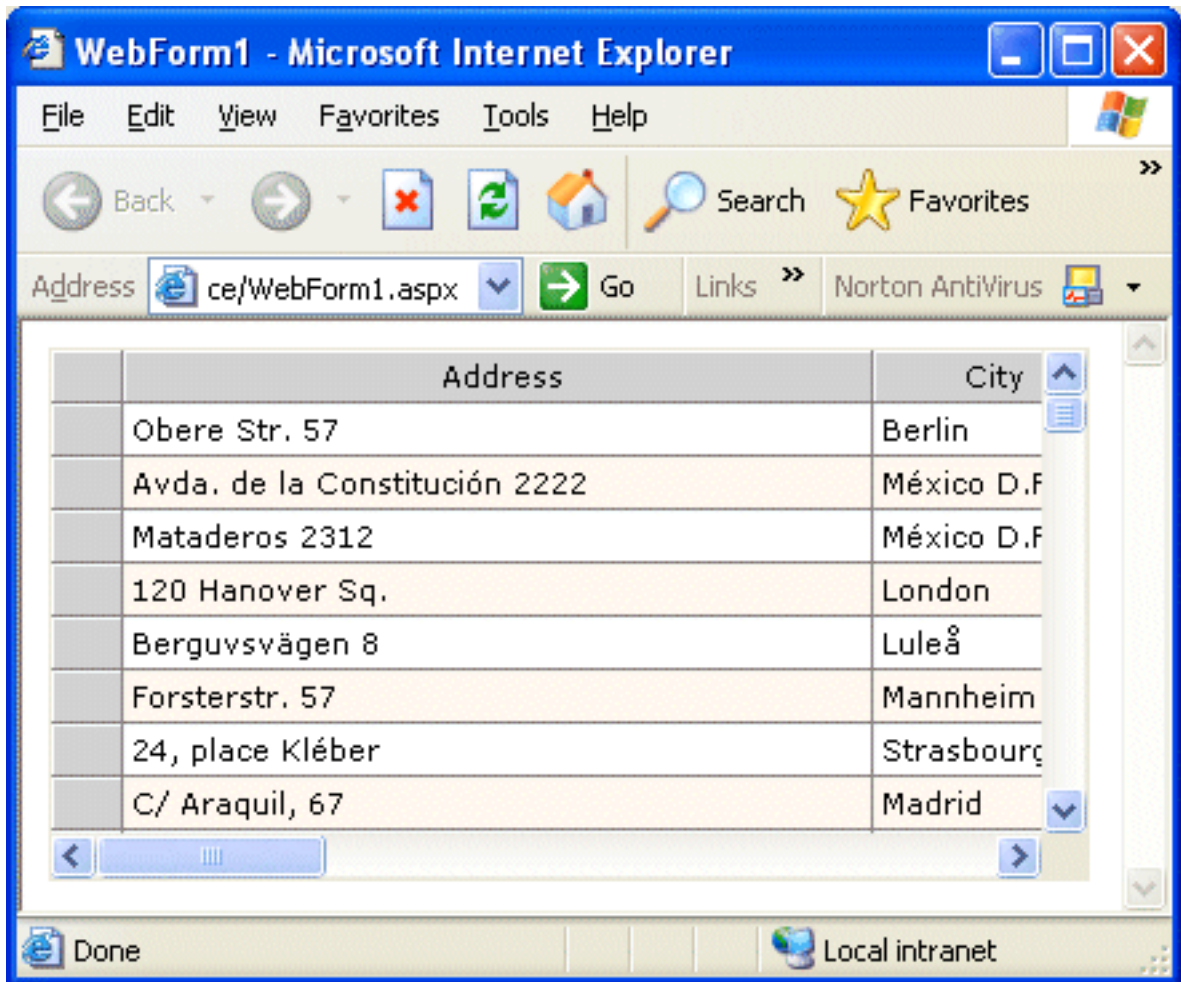
9. Run the project and depending on which version of the Northwind database you connect to you should see something like:



10. Set the row alternate style back color property to whatever color you would like:

- Click on the WebGrid in the designer.
- In Properties, expand DisplayLayout.
- Expand RowAlternateStyleDefault.
- Click the drop-down button for the BackColor property.
- Select a color for the BackColor property.

11. Run the project and you should see something like:



Review

This sample project shows how to change the background color of every-other row by setting the BackColor property of the DisplayLayout.RowAlternateStyleDefault property.

Row Templates

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

There are many times when presenting a user with a grid for data entry is not very user friendly. Especially for users who are expecting forms oriented data entry. Sometimes this problem can be solved by using the grid as a navigator and performing a post back to the server when the user selects a grid row for editing and presenting the user with a new form, but this requires a post back and an additional form.

Row Templates provide a very sophisticated advance in thin client user interface design and usability. The data entry form is a Template and is transported to and interfaces directly with the grid.

Questions

- I am using a WebGrid for user data entry. I would like to offer the user an alternative way of entering data. How can I present the user with a form-style data entry interface without having to do a round trip to the server and present a different form?

Solution

Implement Row Templates. Not only does this technology present the user with a form-style interface with no round trip to the server, but the template layout and server interface is all contained in the web form where the WebGrid resides, thus significantly reduced project complexity.

Sample Project

To create this project, perform the following steps:

1. Start a new Web Forms project.
2. From the Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements
 - Generate the SQL Statements: Click Query Builder, Add Customers and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main menu, select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to `Infragistics.WebUI.Shared`. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.

Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

6. Click on the WebGrid and set the following properties:

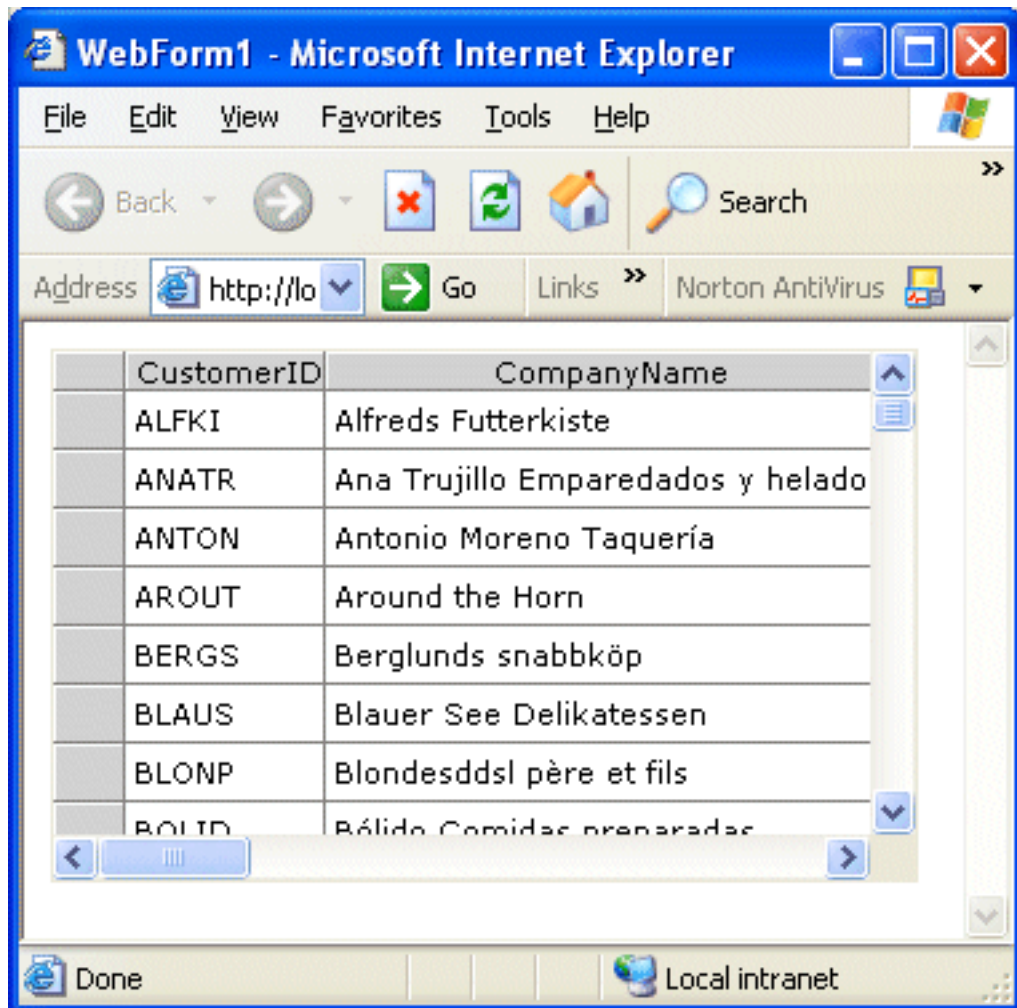
DataSource = DataSet11 **DataMember** = Customers

7. Add the following code to the page's **Load** event:

In Visual Basic:

```
If Me.IsPostBack = False Then
    OleDbDataAdapter1.Fill(DataSet11)
    UltraWebGrid1.DataBind()
End If
```

8. Run the project and you should see something like:



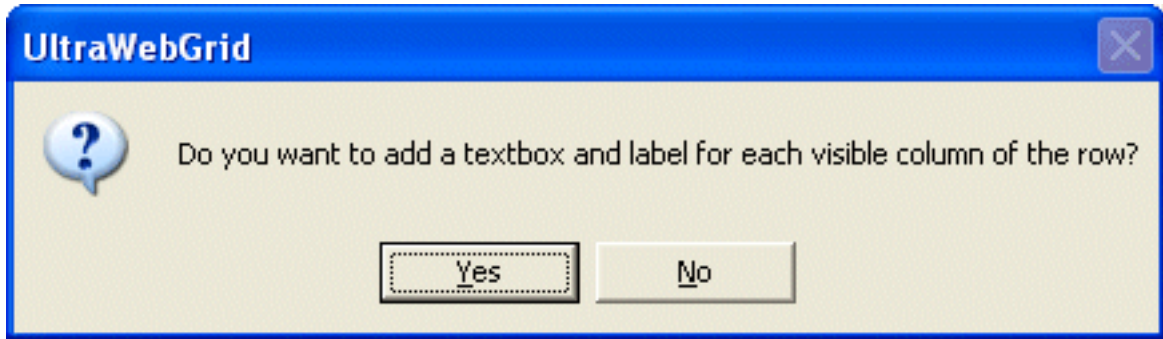
9. On the WebForm1.aspx designer form click on the WebGrid and set the following properties:

Bands(Customers).**AllowUpdate** = RowTemplateOnly

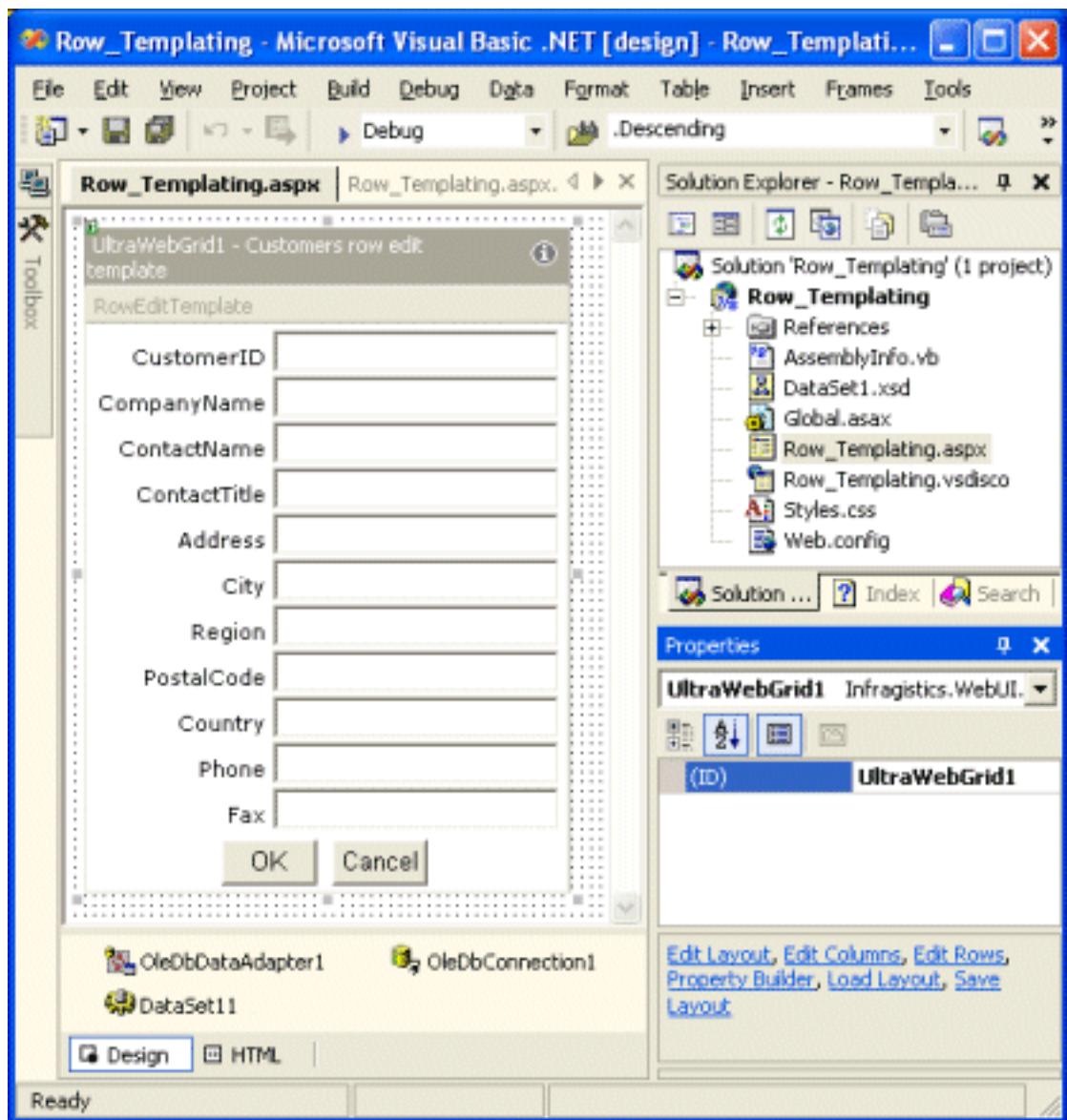
10. Right-click the WebGrid and select Edit Template, Customers Row Edit Template and you should see the following message box:



11. Click Yes and you should see this message box:

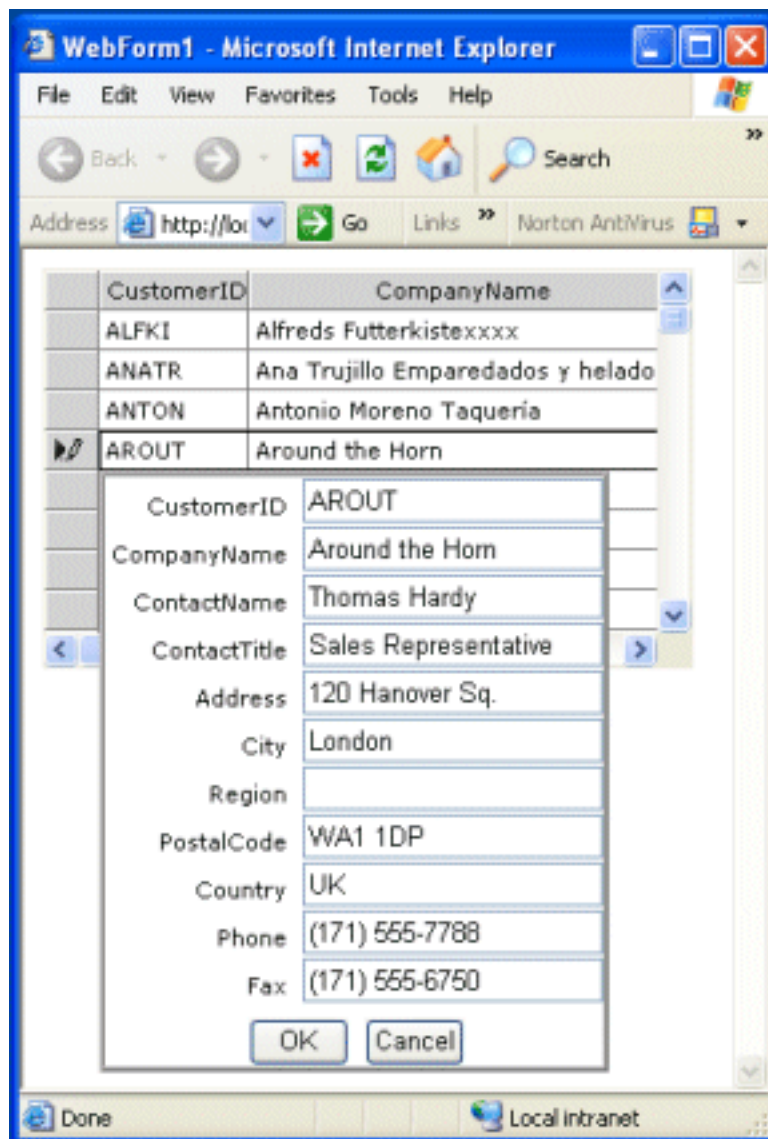


12. Click Yes and you should see the default edit template in the designer. Something like the following:



Notice that the designer automatically added each field to the template.

13. Right-click the template and select End Template Editing to return to the web form designer.
14. Run the project and you should see the original grid something like the first time you ran the project.
15. Click once on the row selector and you will see the row selector image change to indicate the active row being edited.
16. Click on the row selector again and notice the template displays something like:



Notice the default template contains a field for each column in the row. Whenever you change data in the template, the change is transferred back to the WebGrid when the user clicks OK.

Updating the user changes in the database does not change with the use of templates. See the tutorials Update Database with Batch Changes.

Review

This tutorial shows how to quickly setup WebGrid Row Templates to perform user friendly editing of grid data.

Column Appearance

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Many applications require sophisticated column formatting to provide emphasis for important information. The WebGrid provides many different appearance style options on the column. In addition, the developer can also use Column Templates to place any control in the column header, cells and footer.

Questions

- How can I modify the appearance of a single column to emphasize important information?

Solution

Use the styles of the column to modify a columns appearance.

Sample Project

This sample project displays the Northwind Customers database and sets various appearance style properties of the ContactName column. Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Customers and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to `Infragistics.WebUI.Shared`. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
6. Click the Infragistics.WebUI.Shared reference and set the following properties:
CopyLocal = True
7. Click on the WebGrid and set the following properties:

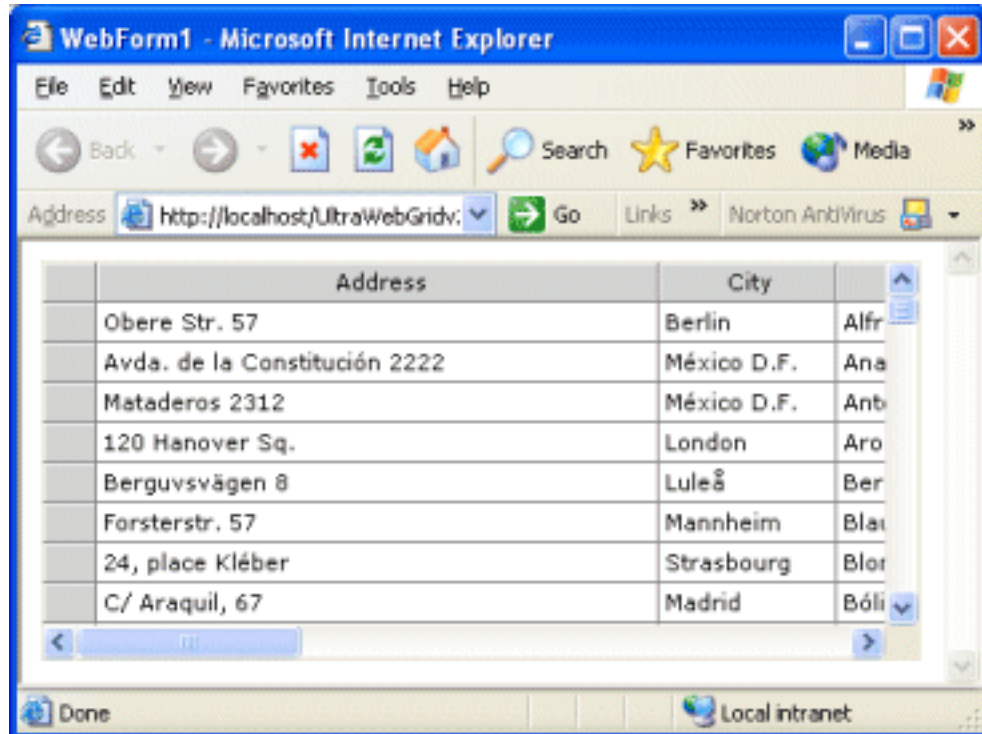
DataSource = DataSet11
DataMember = Customers

8. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then  
    OleDbDataAdapter1.Fill(DataSet11)  
    UltraWebGrid1.DataBind()  
End If
```

9. Run the project and depending on which version of the Northwind database you connect to you should see something like:



Terminate the project.

10. Click on the WebGrid in the designer and from the Properties dialog select the Columns Collection and open the Column Collection Editor.
11. Rearrange the columns into the following order:
1. CustomerID
 2. ContactName
 3. ContactTitle
 4. Phone
 5. Fax
 6. CompanyName
 7. Address
 8. City
 9. Region
 10. PostalCode

11. Country

12. Select the ContactName.

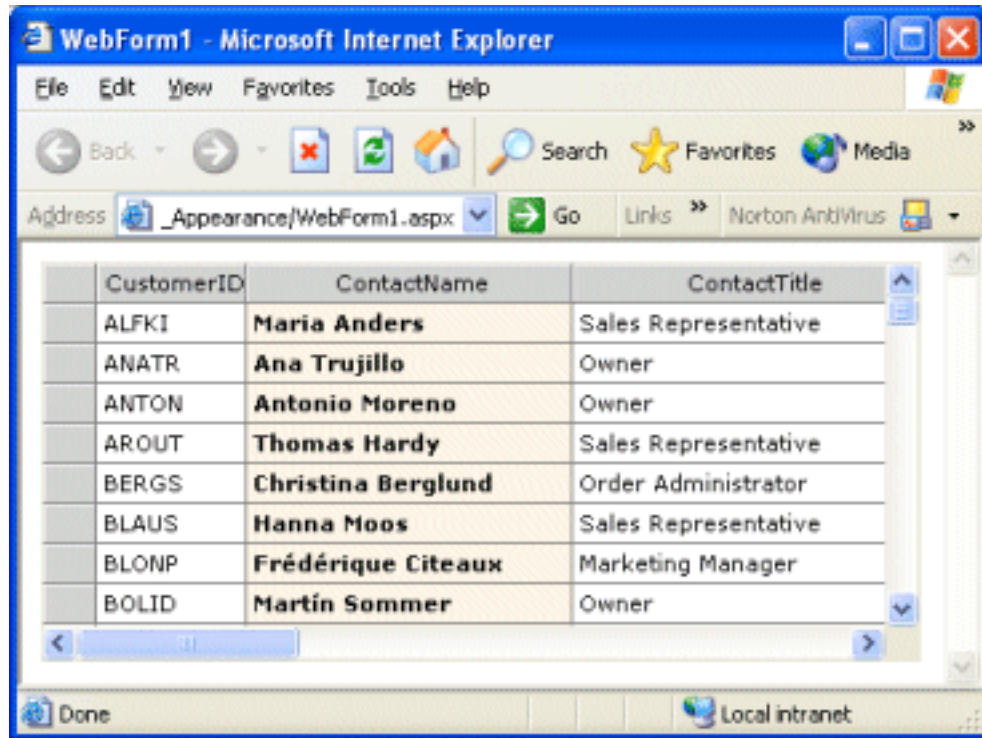
13. Expand CellStyle and set the following properties:

CellStyle.**BackColor** = OldLace

CellStyle.**Font.Bold** = True

CellStyle.**Font.Name** = Verdana

14. Run the project and observe the changes in appearance of the ContactName column:



Terminate the project.

15. Open the Column Collection Editor and set the following properties to enhance the appearance of the ContactName column header:

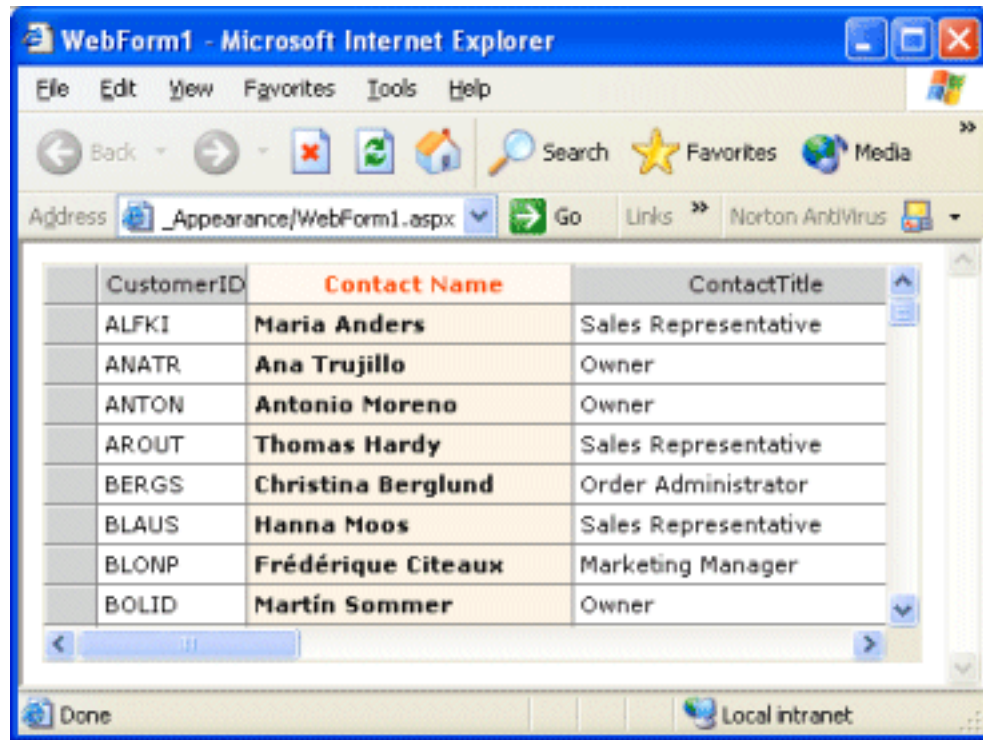
HeaderStyle.**BackColor** = FloralWhite

HeaderStyle.**Font.Bold** = True

HeaderStyle.**ForeColor** = OrangeRed

HeaderText = "Contact Name"

16. Run the project and observe the changes made to the column header appearance.



Review

This tutorial shows how to modify column appearance style properties to highlight important information.

Column Moving

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solutions](#)
- [Sample Project](#)
- [Review](#)

Background

When a WebGrid is bound to a source of data the columns default to the order they are in the columns collection of the underlying data. The developer can move columns through the Column Collections Editor or through code with the **Move** method of the column.

Questions

- How can I use the designer to move columns?
- How do I move columns using code?
- How do I allow the user to move columns?

Solutions

- Use the Column Collection Designer to move columns.
- Use the **Move** method of a column object to move the column using code.
- Set the DisplayLayout.**AllowColMovingDefault** property to OnServer to allow the user to move columns.

Sample Project

This project uses the Northwind Order Details data table as a source of columns to illustrate column moving. Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
4. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.

Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.

5. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

6. Click on the WebGrid and set the following properties:

DataSource = DataSet11

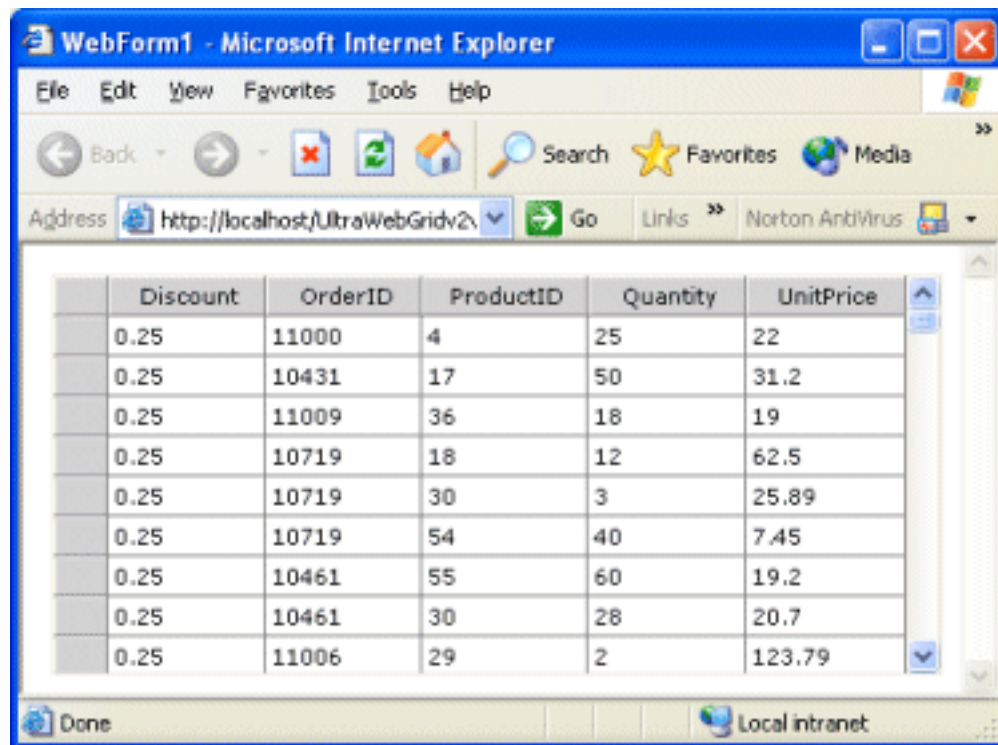
DataMember = Customers

7. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbSelectCommand1.CommandText _
    = "SELECT TOP 100 * FROM [Order Details] ORDER BY Discount DESC"
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

8. Run the project and depending on which version of the Northwind database you connect to you should see something like:



The screenshot shows a Microsoft Internet Explorer browser window titled "WebForm1 - Microsoft Internet Explorer". The address bar shows "http://localhost/UltraWebGridv2". The main content area displays a table with the following data:

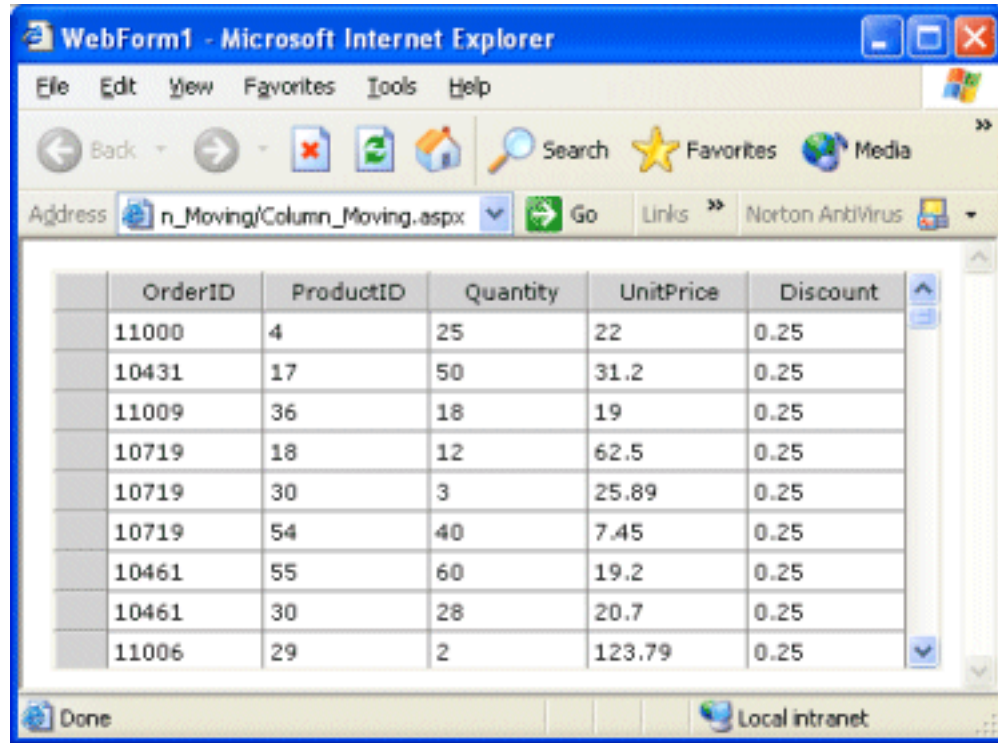
Discount	OrderID	ProductID	Quantity	UnitPrice
0.25	11000	4	25	22
0.25	10431	17	50	31.2
0.25	11009	36	18	19
0.25	10719	18	12	62.5
0.25	10719	30	3	25.89
0.25	10719	54	40	7.45
0.25	10461	55	60	19.2
0.25	10461	30	28	20.7
0.25	11006	29	2	123.79

Terminate the project.

9. Move the Discount column to the right side of the grid following the UnitPrice with the Columns Collection Editor using the following steps:
 - Click the WebGrid on the designer.
 - Click the Columns property and click the ellipse button to open the Columns Collection Editor.
 - Click the down arrow button to move the Discount column to the bottom of the list.

- Click OK.

10. Run the project and observe that the Discount column is now on the right:



OrderID	ProductID	Quantity	UnitPrice	Discount
11000	4	25	22	0.25
10431	17	50	31.2	0.25
11009	36	18	19	0.25
10719	18	12	62.5	0.25
10719	30	3	25.89	0.25
10719	54	40	7.45	0.25
10461	55	60	19.2	0.25
10461	30	28	20.7	0.25
11006	29	2	123.79	0.25

Terminate the project.

11. Move the ProductID column to the left of the OrderID column by adding the following code to the WebGrid InitializeLayout event:

In Visual Basic:

```
' move ProductID to the left side of the grid  
With e.Layout.Bands(0).Columns  
    .FromKey("ProductID").Move(0)  
End With
```

12. Run the project and observe the ProductID column position:

WebForm1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites Media

Address n_Moving/Column_Moving.aspx Go Links Norton AntiVirus

ProductID	OrderID	Quantity	UnitPrice	Discount
4	11000	25	22	0.25
17	10431	50	31.2	0.25
36	11009	18	19	0.25
18	10719	12	62.5	0.25
30	10719	3	25.89	0.25
54	10719	40	7.45	0.25
55	10461	60	19.2	0.25
30	10461	28	20.7	0.25
29	11006	2	123.79	0.25

Done Local intranet

Terminate the project.

13. To allow the user the ability to move columns and make the mouse cursor remain a pointer, perform the following steps:

- Click the WebGrid on the designer.
- In the Properties dialog set the following property:

DisplayLayout.**AllowColMovingDefault** = OnServer
 DisplayLayout.**HeaderStyleDefault.Cursor** = Default

14. Run the project. Click on the Quantity column header and drag the header to the left and drop it between the ProductID and OrderID columns when the column move target indicators display:

WebForm1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

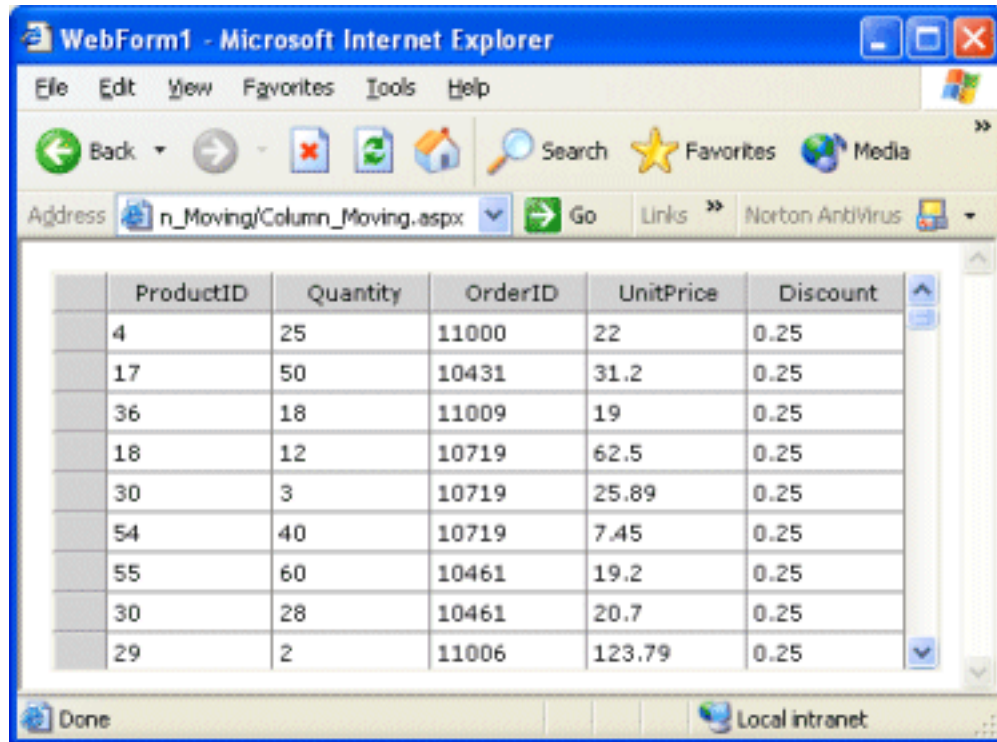
Back Forward Stop Refresh Home Search Favorites Media

Address n_Moving/Column_Moving.aspx Go Links Norton AntiVirus

ProductID	Quantity	OrderID	Quantity	UnitPrice	Discount
4	11000	25	22	0.25	
17	10431	50	31.2	0.25	
36	11009	18	19	0.25	
18	10719	12	62.5	0.25	
30	10719	3	25.89	0.25	
54	10719	40	7.45	0.25	
55	10461	60	19.2	0.25	
30	10461	28	20.7	0.25	
29	11006	2	123.79	0.25	

Done Local intranet

15. The WebGrid will perform a post back to the server and the column will be moved:



Terminate the project.

Review

This tutorial shows three different ways to move columns: (1) Using the Column Collection Editor, (2) Using the Move method of the column object, and (3) allowing the user to move the columns.

Stationary Column Headers

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

By default when the grid is scrolled the column headers scroll with the rows. WebGrid provides a property that makes the column headers stationary so they do not scroll with the rows.

The WebGrid DisplayLayout.**StationaryMargins** property supports stationary headers, footers, or both headers and footers.

Questions

- How can I make the column headers stay visible when the user scrolls the rows?

Solution

Set the DisplayLayout.**StationaryMargins** property to Header.

Sample Project

This project uses the Northwind Orders database as a source of data to illustrate the use of stationary column headers.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
4. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
5. Click the Infragistics.WebUI.Shared reference and set the following properties:
CopyLocal = True

6. Click on the WebGrid and set the following properties:

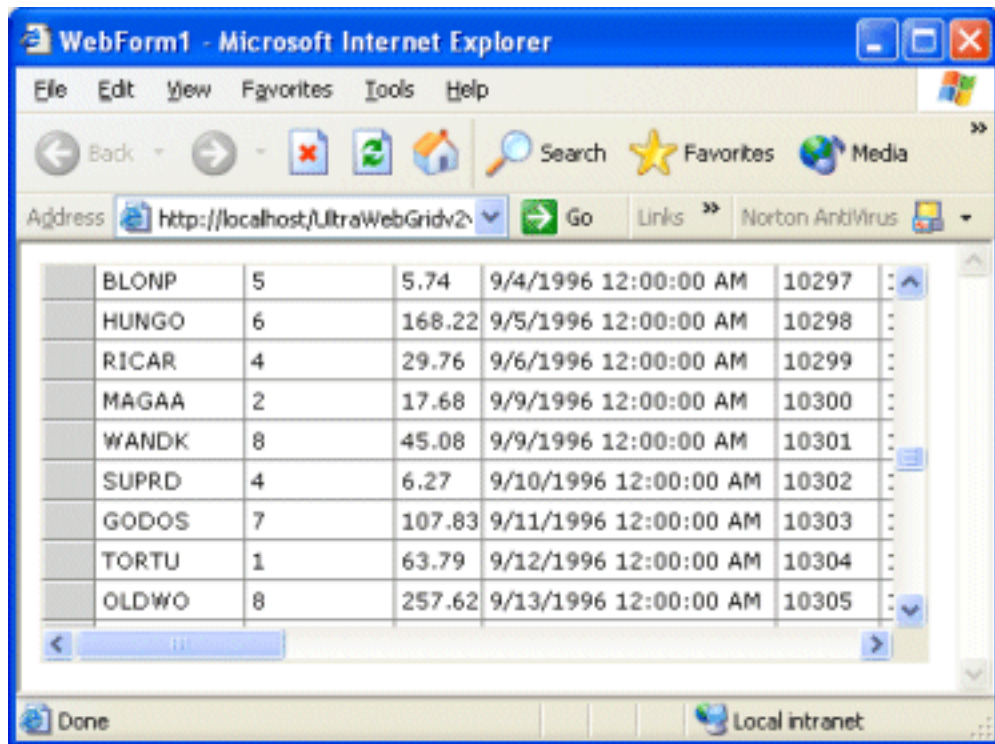
DataSource = DataSet11
DataMember = Customers

7. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbSelectCommand1.CommandText _
    = "SELECT TOP 100 * FROM [Order Details] ORDER BY Discount DESC"
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

8. Run the project. Click on the scroll bar drag button and drag it about half way towards the bottom. Observe the column headers scroll with the rows:

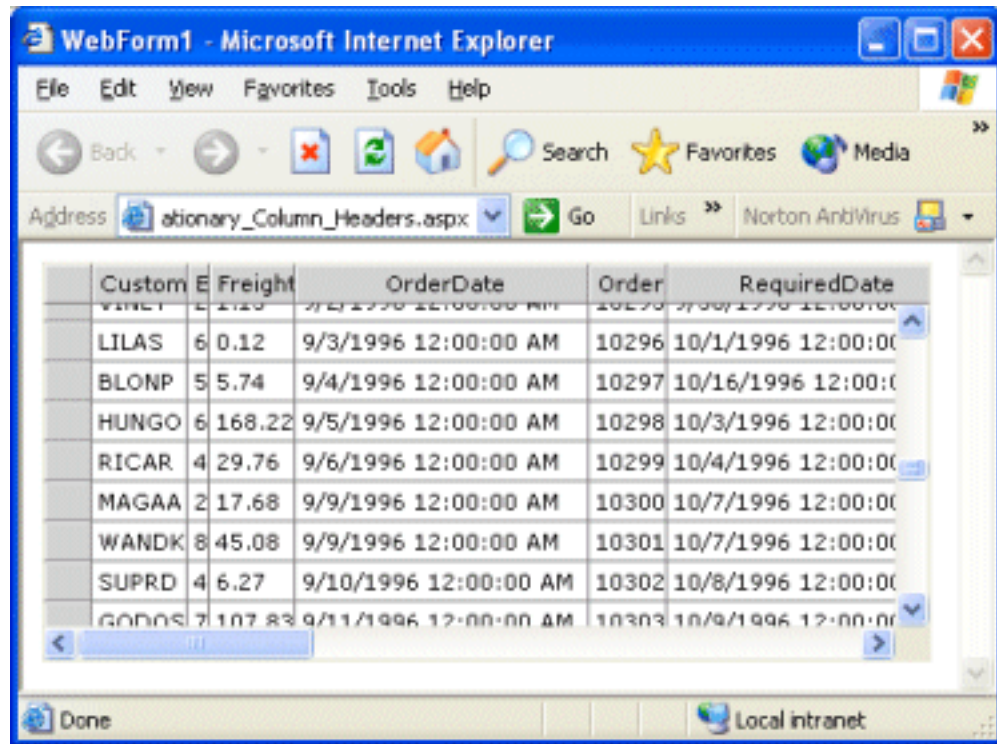


Terminate the project.

9. Click on the WebGrid on the WebForm1.aspx designer and in the Properties Dialog set the following property:

DisplayLayout.**StationaryMargins** = Header

10. Run the project, drag the scroll button down the form and observe the headers remain visible:



The **StationaryMargins** property can make the Headers, Footers or both stationary.

Review

This tutorial shows how to make the column headers stationary rather than scrolling with the rows.

Column Sorting and Sort Indicators

There are 6 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Code Discussion](#)
- [Review](#)

Background

When the built-in column sort behavior does not meet the needs of your application, this behavior can be overridden in the SortColumn event by repopulating the grid and setting e.Cancel to True.

One scenario where this is helpful is in an application where thousands of records can be selected and these records are paged to the control. In this case there are only 10 to 25 rows actually bound to the grid at any one time. The built-in column sorting will sort the bound rows, but cant sort the thousands of rows and reorganize the paging.

Questions

- How can I override the built-in sort behavior of the column header click?
- I have an application that requires thousands of records display in pages. How can I use the column header click to sort the underlying records?

Solution

The built-in sort behavior is overridden in the **SortColumn** event by performing whatever action is desired on the grid and setting the e.Cancel property to True. Setting the e.Cancel property to True cancels the built-in behavior.

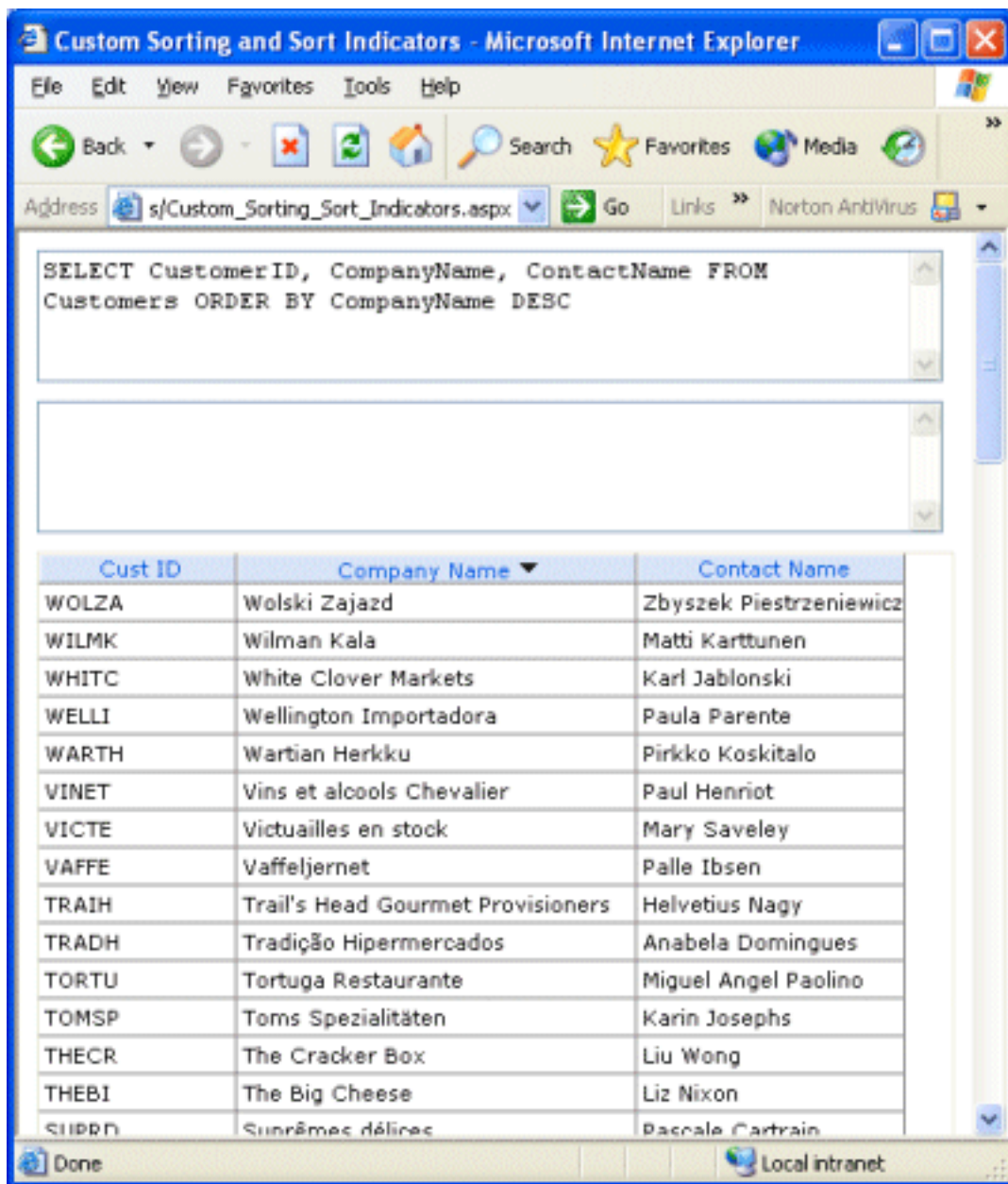
For the case where thousands of records may be included in the underlying selection, query the database depending upon the column clicked and the previous sort order of the column and bind a new page of records to the grid. See the topic titled *Custom Selection and Paging* for a complete discussion of these methodologies.

Sample Project

This sample project queries the database based on the column clicked and the previous sort order of the column and binds the results of the new query to the grid. The Sort Indicators are set to reflect the type of query performed. The Northwind Customers database is used and the sample project displays the SQL statement in a text box.

Note This tutorial requires the `uwg2_Helper.vb` module. You can obtain this file by right-clicking on [this link](#) and choosing "Save Target As..." from the context menu. When the File Save dialog appears, save the file in the same folder as the other files that make up the project. You should then add this file to your project.

Run the project and click on the column headers to exercise the custom sorting and observe the sort indicators:



Code Discussion

Files containing code in this project:

Uwg2_Helper.vb

This code is not reviewed in this exercise.

Custom_Sorting_Sort_Indicators.aspx

Custom_Sorting_Sort_Indicators.aspx contains the code relevant to this project and consists of the following code regions:

Class Declarations

The Class Declarations Region contains declarations local to the Class:

Declare a connection object for connecting to the database and variables for sharing of the last Sort Column Name and Sort Ascending flag:

In Visual Basic:

```
Dim c_cnDatabase As New OleDb.OleDbConnection()  
Dim c_blnSortColumnName As String = ""  
Dim c_blnSortAscending As Boolean = True
```

Private Methods

The Private Methods Region contains the following methods:

- **PerformSearch** - The **PerformSearch** method performs the following tasks:
 - Saves the sort request details (Sort Column Name and Sort Ascending Flag) in session variables so on a post back the previous values can be retrieved.
 - Creates an SQL select statement and retrieves the records sorted in the requested order. This puts the burden of sorting the records on the SQL Server. This can be a significant task when thousands of records are selected.
 - Binds the selected records to the grid.

In Visual Basic:

```
Private Sub PerformSearch(ByVal v_strOrderByColumnName As String _  
    , ByVal v_blnSortAscending As Boolean)  
  
    Dim sbSQL As New System.Text.StringBuilder(4096)  
    Dim dtSelected As DataTable  
  
    ' save sort details  
    c_blnSortColumnName = v_strOrderByColumnName  
    c_blnSortAscending = v_blnSortAscending  
    Session("SortColumnName") = c_blnSortColumnName  
    Session("SortAscending") = c_blnSortAscending  
  
    ' build select part of SQL  
    sbSQL.Append("SELECT CustomerID, CompanyName" _  
    + ", ContactName FROM Customers ORDER BY " + c_blnSortColumnName)  
  
    ' test for descending sort  
    If v_blnSortAscending = False Then  
        sbSQL.Append(" DESC")  
    End If  
  
    ' put SQL into display  
    txtSQL.Text = sbSQL.ToString  
    Try  
        ' build data adapter  
        Dim daSelected As OleDb.OleDbDataAdapter _  
            = UWG2.Utility.OleDbDataAdapter.CreateSelectOnly( _  
            sbSQL.ToString, c_cnDatabase)  
  
        ' make new data table and retrieve records  
        dtSelected = New DataTable()  
        daSelected.MissingSchemaAction = MissingSchemaAction.AddWithKey  
        daSelected.Fill(dtSelected)  
  
    Catch ex As Exception  
        txtMessage.Text = ex.Message  
    End Try  
  
    ' bind table to grid  
    UltraWebGrid1.DataSource = dtSelected  
    UltraWebGrid1.DataBind()  
End Sub
```


Page Events

The Page Events Region contains the following methods:

- **Page_Load** (*Handles MyBase.Load*) - The **Page_Load** method retrieve a connection string to the database, retrieves some values from session variables, and if this is not a post back, performs the initial search:

In Visual Basic:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _  
  
    Handles MyBase.Load  
    'Put user code to initialize the page here  
    c_cnDatabase.ConnectionString = UWG2.Utility.DataConnection.ConnectionString()  
    c_blnSortColumnName = CStr(Session("SortColumnName"))  
    c_blnSortAscending = CType(Session("SortAscending"), Boolean)  
  
    If Me.IsPostBack = False Then  
        PerformSearch("CompanyName", True)  
    End If  
End Sub
```

UltraWinGrid Events

The UltraWinGrid Events Region contains the following methods:

- **UltraWebGrid1_InitializeLayout** (*Handles UltraWebGrid1.InitializeLayout*) - The InitializeLayout event contains the code used to format the grid and set the column heading sort indicator:

In Visual Basic:

```
Private Sub UltraWebGrid1_InitializeLayout( _  
    ByVal sender As Object, ByVal e As Infragistics.WebUI.UltraWebGrid.LayoutEventArgs) _  
    Handles UltraWebGrid1.InitializeLayout  
  
    With e.Layout.Bands(0)  
        ' set data key field  
        .DataKeyField = "CustomerID"  
  
        ' set header style  
        .HeaderStyle.BackColor = Color.FromName("InactiveCaptionText")  
        .HeaderStyle.ForeColor = Color.FromName("ActiveCaption")  
        .HeaderStyle.Cursor = Infragistics.WebUI.UltraWebGrid.Cursors.Default  
  
        ' set column headings  
        .Columns.FromKey("CustomerID").HeaderText = "Cust ID"  
        .Columns.FromKey("CompanyName").HeaderText = "Company Name"  
        .Columns.FromKey("ContactName").HeaderText = "Contact Name"  
  
        ' clear all column header sort indicators  
        Dim uwgcol As Infragistics.WebUI.UltraWebGrid.UltraGridColumn  
        For Each uwgcol In .Columns  
            If uwgcol.SortIndicator = _  
                <> Infragistics.WebUI.UltraWebGrid.SortIndicator.None Then  
                uwgcol.SortIndicator = _  
                    Infragistics.WebUI.UltraWebGrid.SortIndicator.None  
            End If  
        Next  
  
        ' set column heading sort indicator
```

```

If c_blnSortColumnName <> "" Then
    If c_blnSortAscending = True Then
        .Columns.FromKey(c_blnSortColumnName).SortIndicator _
        = Infragistics.WebUI.UltraWebGrid.SortIndicator.Ascending
    Else
        .Columns.FromKey(c_blnSortColumnName).SortIndicator _
        = Infragistics.WebUI.UltraWebGrid.SortIndicator.Descending
    End If
End If

' set other options
.RowSelectors _
= Infragistics.WebUI.UltraWebGrid.RowSelectors.No
.CellClickAction _
= Infragistics.WebUI.UltraWebGrid.CellClickAction.RowSelect
.AllowSorting _
= Infragistics.WebUI.UltraWebGrid.AllowSorting.Yes
.HeaderClickAction _
= Infragistics.WebUI.UltraWebGrid.HeaderClickAction.SortSingle
End With
End Sub

```

- **UltraWebGrid1_SortColumn** (*Handles UltraWebGrid1.SortColumn*) - The SortColumns event invokes the PerformSearch method which retrieves a new data table from the database and binds it to the grid. The most significant line of code in this event is e.Cancel = True which cancels the built-in sort methods.

In Visual Basic:

```

Private Sub UltraWebGrid1_SortColumn( _
ByVal sender As Object _
, ByVal e As Infragistics.WebUI.UltraWebGrid.SortColumnEventArgs) _
Handles UltraWebGrid1.SortColumn

' perform search for selected column header

Select Case e.BandNo
Case 0
    If c_blnSortColumnName _
= UltraWebGrid1.Bands(e.BandNo).Columns(e.ColumnNo).Key Then
        If c_blnSortAscending = True Then
            PerformSearch(UltraWebGrid1.Bands(e.BandNo).Columns(e.ColumnNo).Key _
, False)
        Else
            PerformSearch(UltraWebGrid1.Bands(e.BandNo).Columns(e.ColumnNo).Key _
, True)
        End If
    Else
        PerformSearch(UltraWebGrid1.Bands(e.BandNo).Columns(e.ColumnNo).Key _
, True)
    End If
    e.Cancel = True
End Select
End Sub

```

Review

This tutorial shows how to override the built-in column header click sort methods and replace them with methods that query the database and let the database manager sort the records.

Column Templates

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

There are times when the application requires the default formatting of the column header, column cells and column footer needs to be completely overridden. WebGrid provides Column Templates which allow the developer to completely override all aspects of the column.

Questions

- How can I override the behavior of the column header, column cells and column footer?

Solution

Implement Column Templates to override the behavior of the column header, cells and footer.

Sample Project

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Customers and click Close, click (All Columns) and click OK.Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
6. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

7. Click on the WebGrid and set the following properties:

DataSource = DataSet11

DataMember = Customers

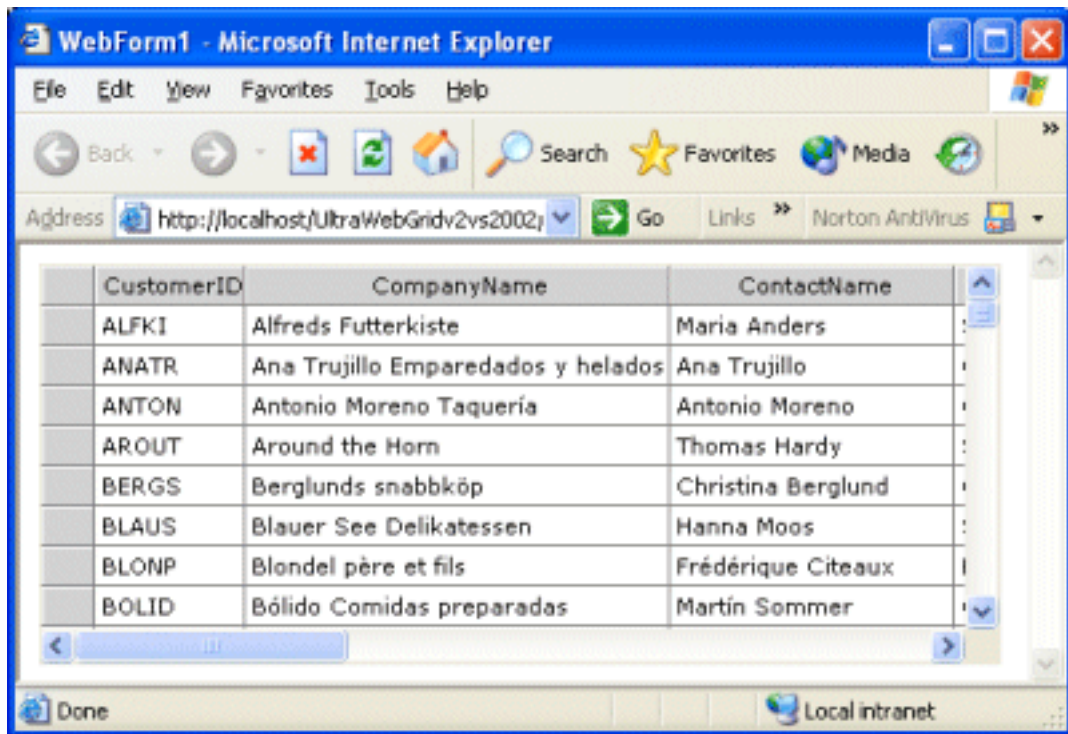
DisplayLayout.ColFootersVisibleDefault = Yes

8. Right-click the grid and select the Edit Columns item. Make the CustomerID column templated by checking the Templated Column check box in the bottom part of the dialog while the column is selected. Close the dialog by hitting the OK button.
9. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

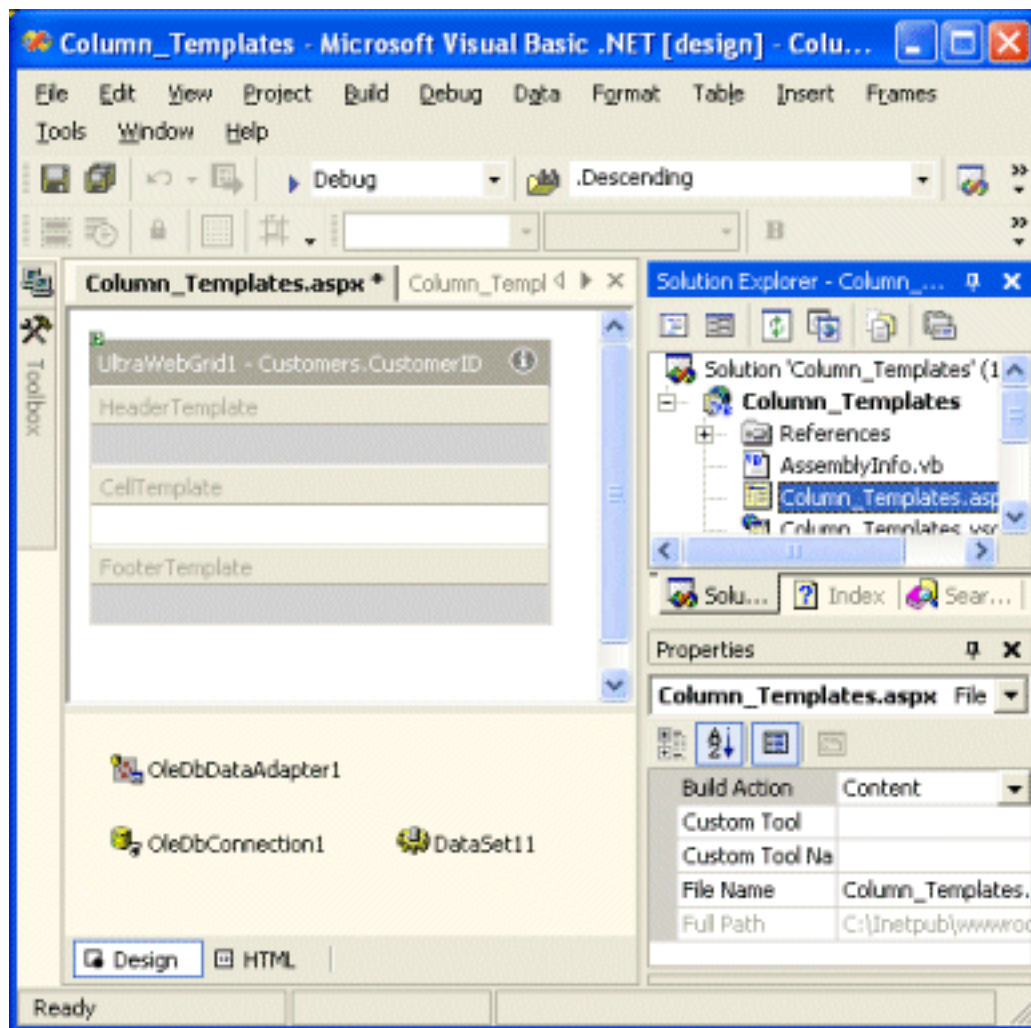
In Visual Basic:

```
If Me.IsPostBack = False Then  
    OleDbDataAdapter1.Fill(DataSet11)  
    UltraWebGrid1.DataBind()  
End If
```

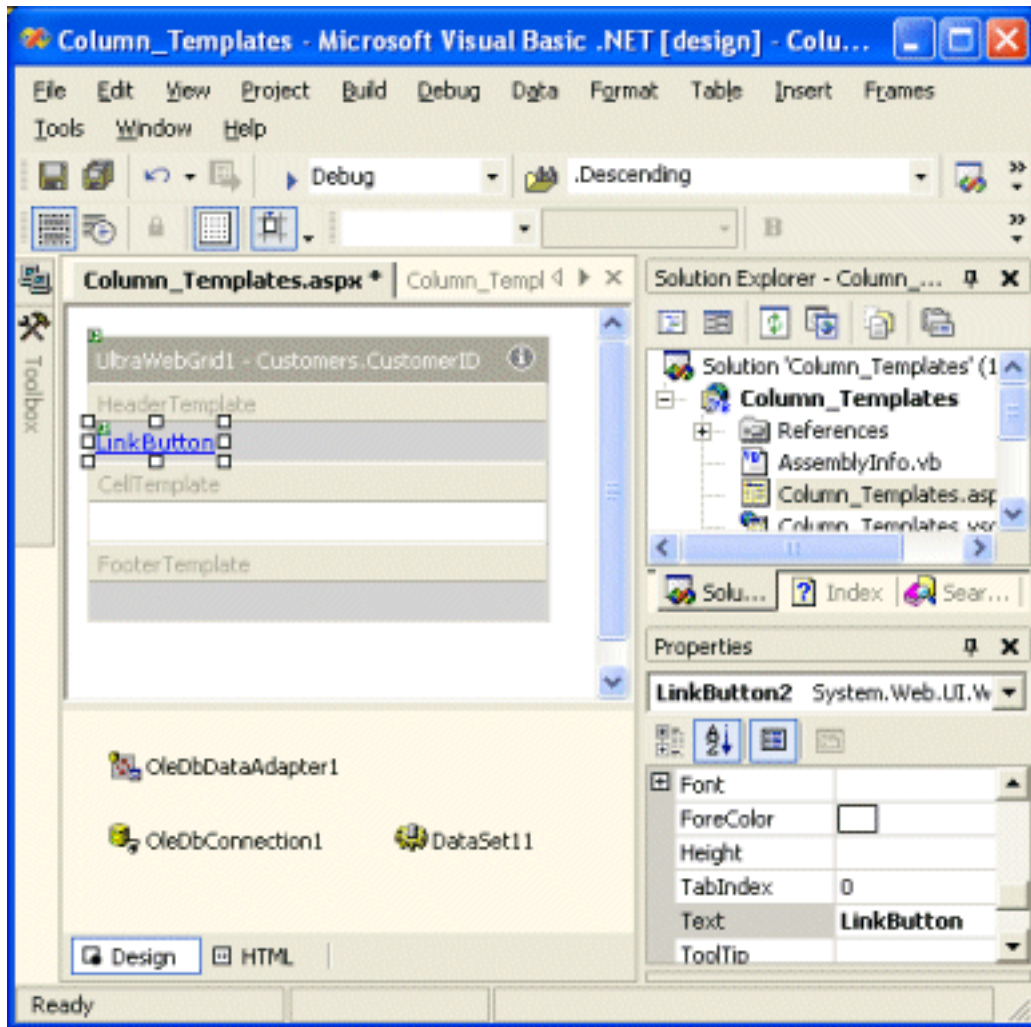
The application should look something like this:



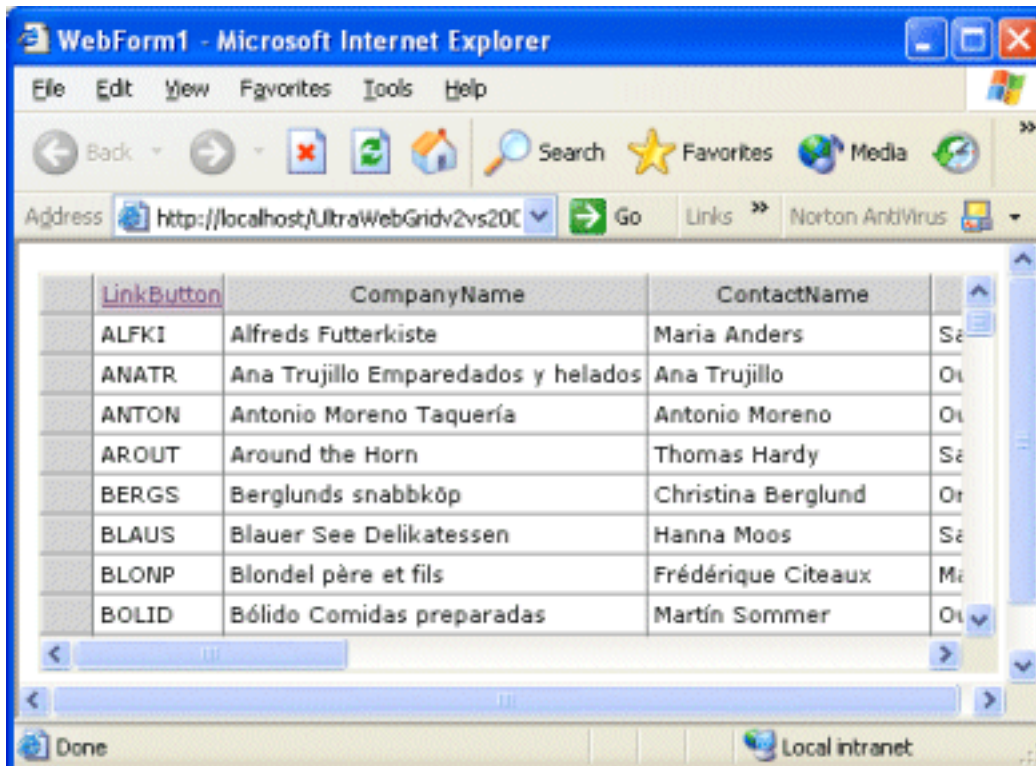
10. To illustrate the use of column templates, place a Link Button in the CustomerID Column header. Right-click on the WebGrid and select Edit Template, Customers.CustomerID and the column template designer displays something like:



Notice in the above illustration the titles Header Template, Cell Template and Footer Template. These are the titles above the area where you will place the control to be rendered in the Header, Cell and Footer of the column. So, to put a Link Button in the header of the column, from Toolbox, Web Forms, drag a Link Button into the gray area below the Header Template title. Now the designer should look something like:



11. Run the project and you will see something like:



12. To show how the footer will look containing a Hyperlink, from Toolbox, Web Forms, drag a Hyperlink

control to the Footer Template.

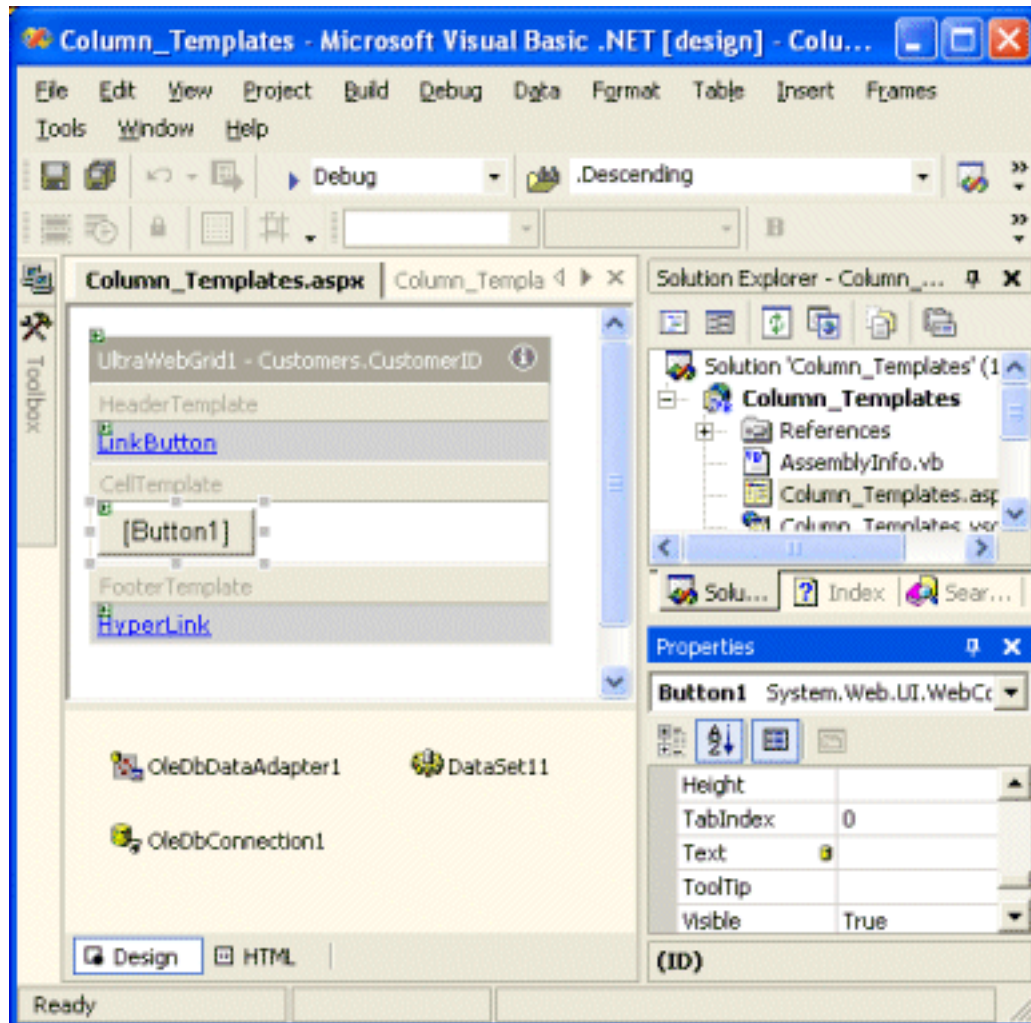
The really nice feature of the Column Template Editor is that you can click on the control and set properties in the Properties pane. This allows you to enter the hyperlink target URL and other properties using designers rather than having to write HTML.

Click on the Hyperlink control and set the following properties:

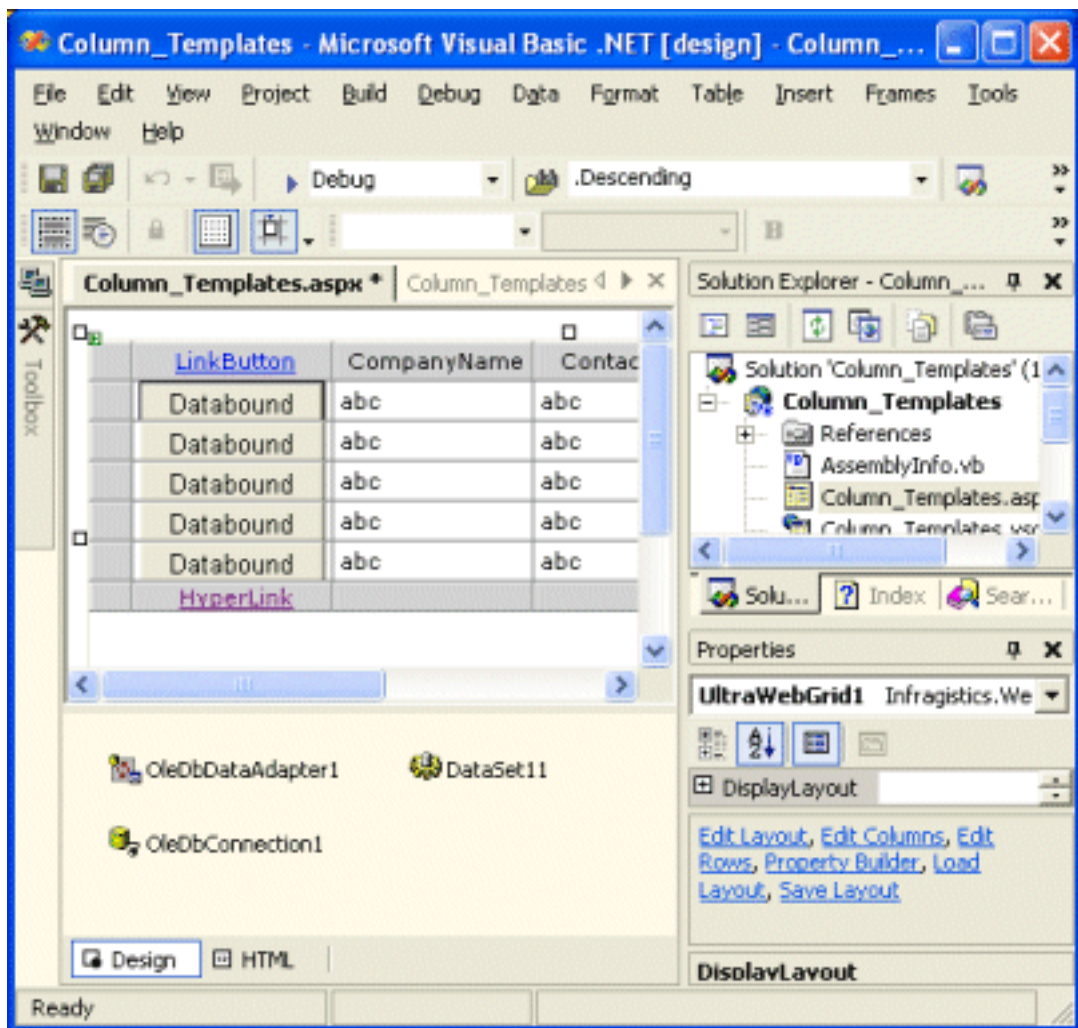
NavigateUrl = <http://www.infragistics.com>

13. To illustrate the binding of data to a template control, place a Button control in the Cell Template. From Toolbox, Web Forms, drag a Button control to the Cell Template. To bind the button text to the underlying container: click on the Button control in the cell template and set the text property to `<% # Container.Text %>`.

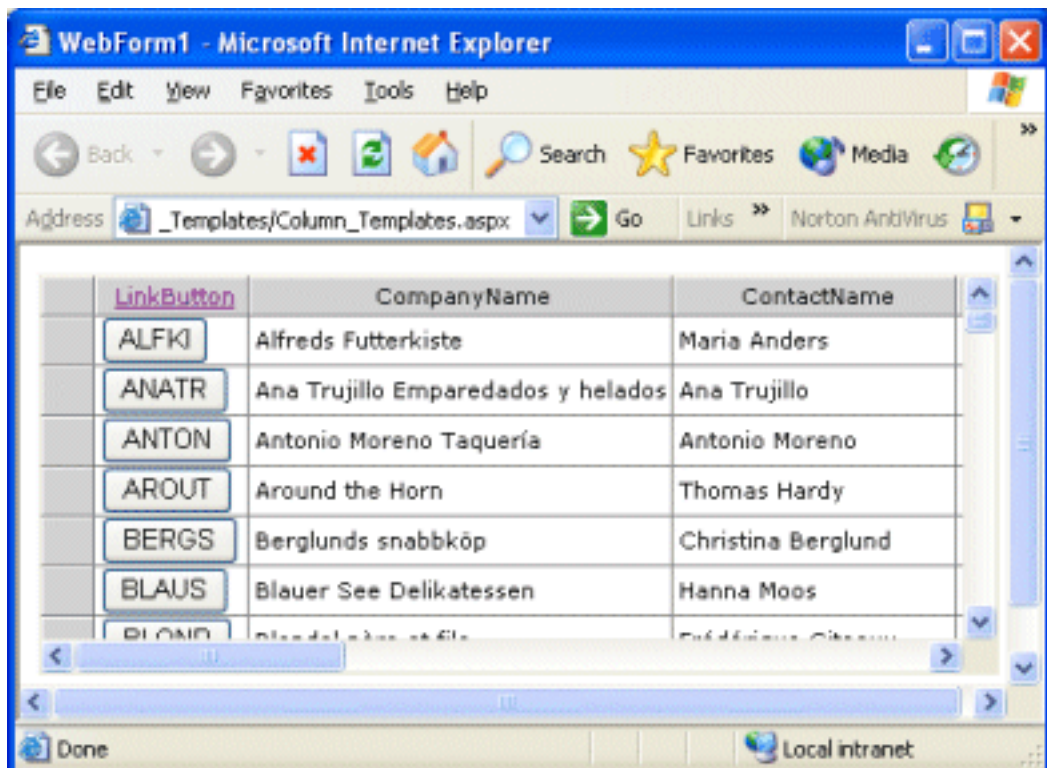
The column template editor should now look something like:



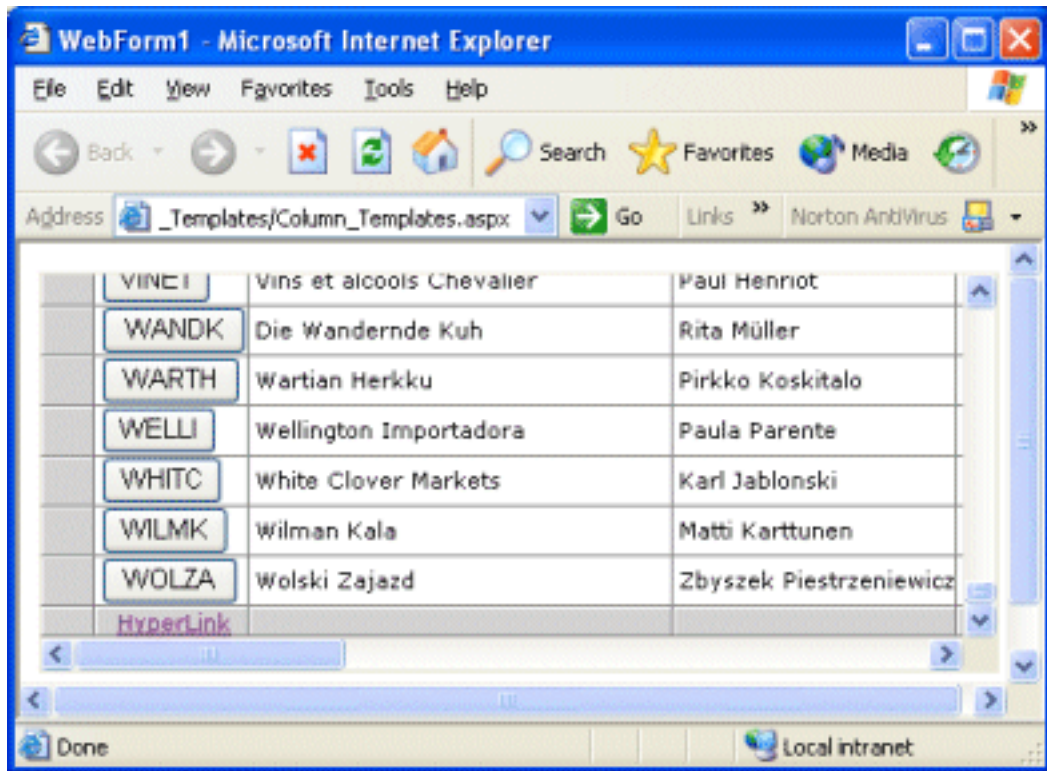
14. To exit the column template editor, right-click the column template designer and select End Template Editing. Visual Studio should return to the Web Form designer and it should display something like:



15. Run the project and you should see something like:



16. Scroll down to the bottom of the grid rows and notice the hyperlink control displayed in the footer:



Review

This tutorial shows the developer how to use the Column Template designer to override the normal grid behavior and apply custom controls to column headers, cells and footers.

Cell Appearance

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

When working with the WebGrid, one of the first appearance modifications requested by users is the appearance of the grid cells.

Questions

- How do I change the appearance of the WebGrid cells?

Solution

Use the properties editors to change the appearance style of cells in the WebGrid. (Also see the tutorials [Grid Appearance Using Styles](#) and [Initialize Row](#) for more information on how to modify cell appearance.)

Sample Project

This sample project displays the Northwind Order Details table and shows how to modify the appearance and format of the cells.

Create this project by performing the following steps:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
6. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

7. Click on the WebGrid and set the following properties:

DataSource = DataSet11
DataMember = Customers

8. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

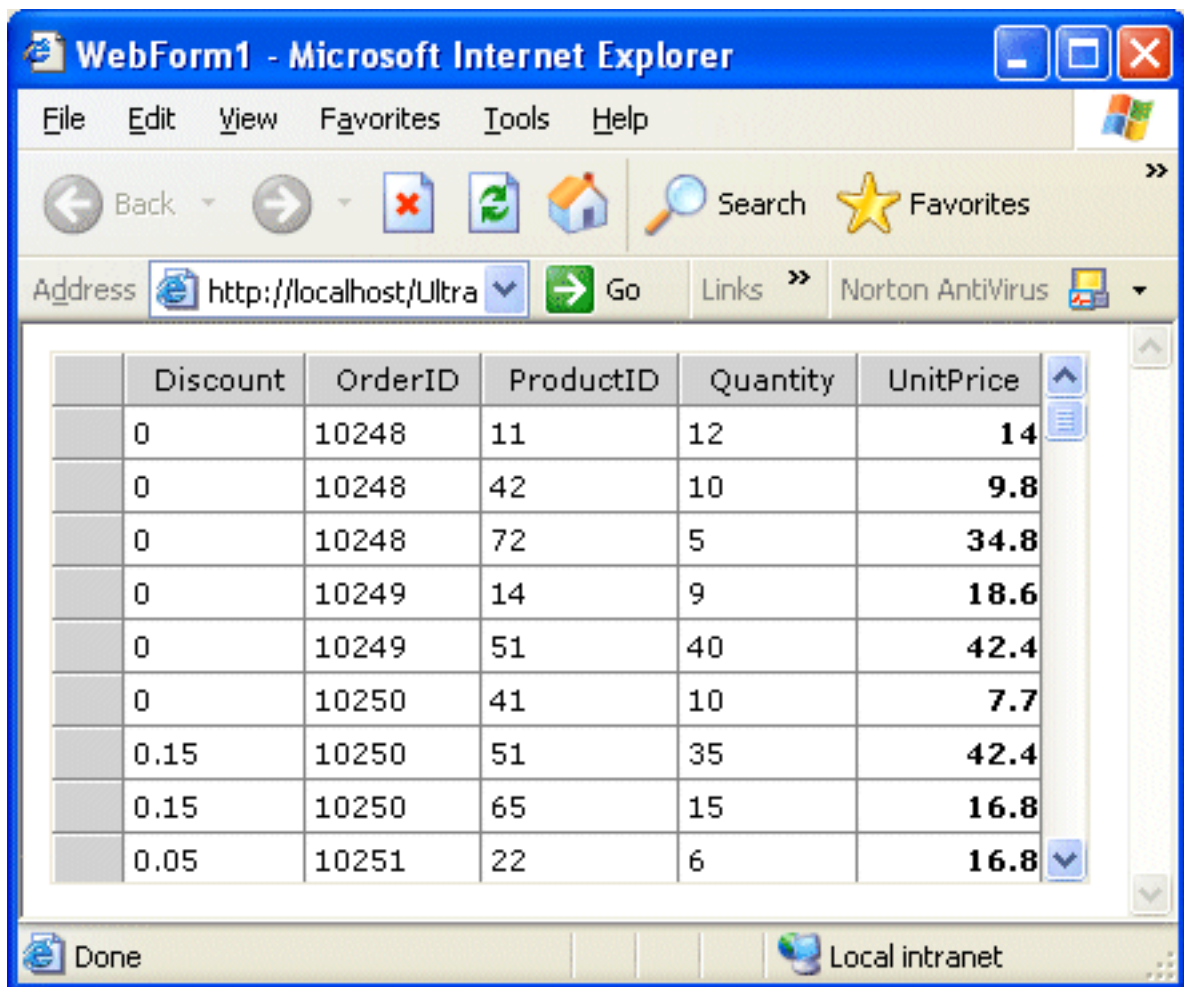
In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbSelectCommand1.CommandText _
    = "SELECT TOP 100 * FROM [Order Details]"
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

9. Change the cells in the UnitPrice column to display in bold and align right:

- Click the WebGrid on the WebForm1.aspx designer.
- In Properties, Click the Columns property.
- Click the ellipse button to open the Columns Collection dialog, then:
- Click the UnitPrice column.
- In UnitPrice Properties, expand the CellStyle node.
- Expand the Font node.
- Set the Bold property to true.
- Set the HorizontalAlign property to true.
- Click OK.

10. Run the project and you should see something like:



Use these steps to change other properties of the cells in the columns. Also notice there are four style settings for a column:

- Cell Button Style
- Cell Style
- Selected Cell Style
- Selected Header Style

Review

This tutorial shows how to set WebGrid cell appearance style properties of a strongly typed dataset using the properties editors.

Cell Merging

There are 6 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Code Discussion](#)
- [Review](#)

Background

HTML tables have always had the ability to merge cells both horizontally and vertically. The WebGrid also has the ability to merge cells as the data is being bound by simply setting the **MergeCells** property to true in the **InitializeLayout** event.

Questions

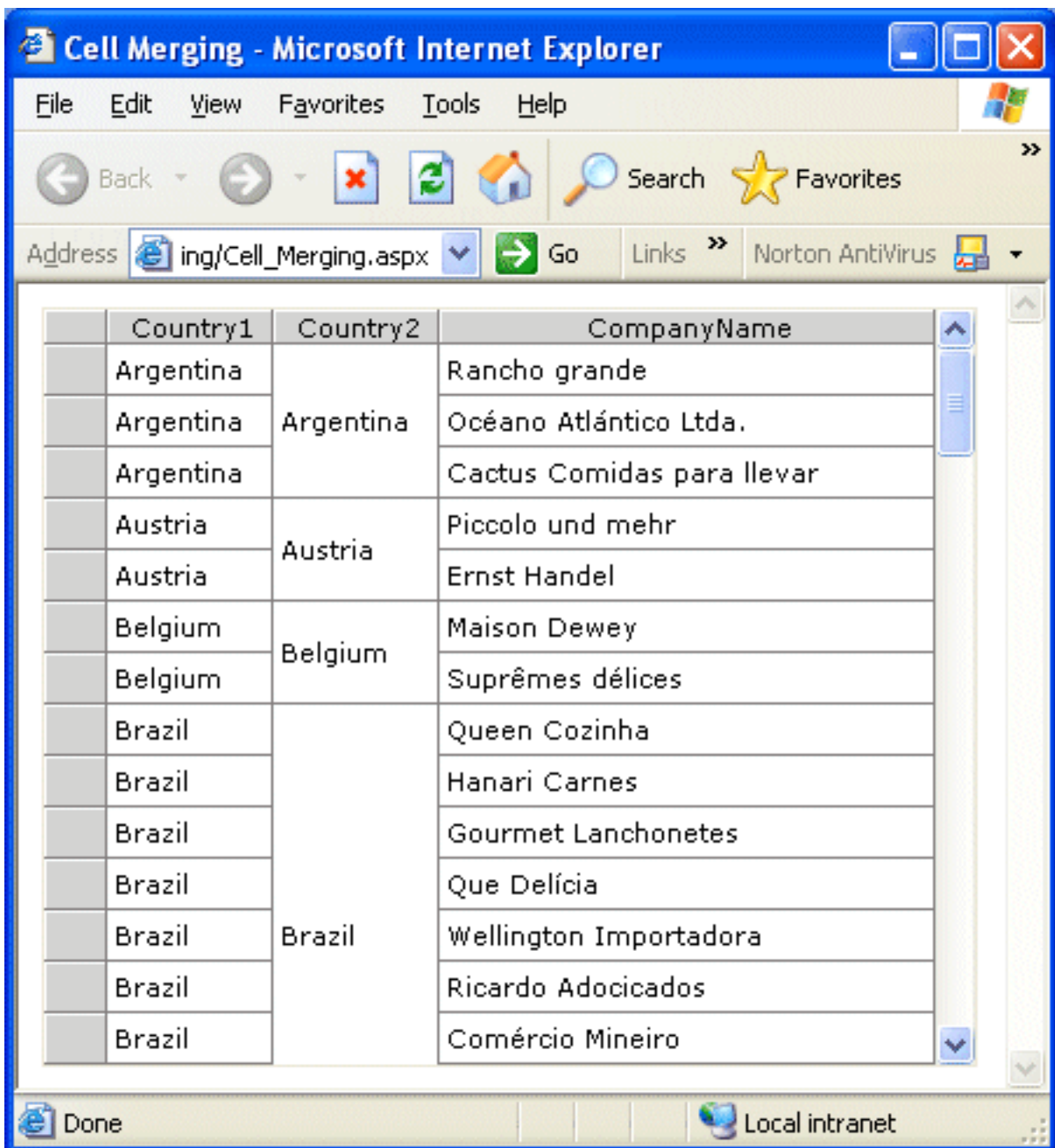
- How can I get the WebGrid to automatically merge cells of the same value in a column like in a classical HTML grid?

Solution

Set the **MergeCells** property of the column to true in the **InitializeLayout** event.

Sample Project

This sample projects uses the Northwind Customers database to illustrate the use of the **MergeCells** property of WebGrid. Notice the Country column displays twice as Country1 and Country2 with the Country2 column having its cells merged.



Note This tutorial requires the `Uwg2_Helper.vb` module. You can obtain this file by right-clicking on [this link](#) and choosing "Save Target As..." from the context menu. When the File Save dialog appears, save the file in the same folder as the other files that make up the project. You should then add this file to your project.

Code Discussion

Files containing code in this project:

Uwg2_Helper.vb

This code is not reviewed in this exercise.

Cell_Merging.aspx

Cell_Merging.aspx contains the code relevant to this project and consists of the following code regions:

Page Events

The Page Events region consists of the following event handlers:

- **Page_Load Handles MyBase.Load** - The **Page_Load** method retrieves a connection string to the

database, creates a data adapter with the appropriate SQL select statement, creates a new data table and fills it with data from the data adapter and binds the data table to the WebGrid:

In Visual Basic:

```
' create a data connection
Dim cnNorthwind As New OleDb.OleDbConnection()
cnNorthwind.ConnectionString = _
UWG2.Utility.DataConnection.ConnectionString()

' create a data adapter
Dim daNorthwindCustomers As OleDb.OleDbDataAdapter = _
UWG2.Utility.OleDbDataAdapter.CreateSelectOnly( _
"SELECT Country as Country1, Country as Country2" _
+ ", CompanyName FROM Customers ORDER BY Country", cnNorthwind)

' create and populate a data table
Dim dtNorthwindCustomers As New DataTable()
daNorthwindCustomers.Fill(dtNorthwindCustomers)

' bind data table to grid
UltraWebGrid1.DataSource = dtNorthwindCustomers
UltraWebGrid1.DataBind()
```

WebGrid Events

The WebGrid Events region consists of the following event handlers:

- **UltraWebGrid1_InitializeLayout Handles UltraWegBrid1.InitializeLayout** - The `UltraWebGrid1_InitializeLayout` method tells the WebGrid to merge cells for the Country2 column:

In Visual Basic:

```
With e.Layout.Bands(0)
.Columns.FromKey("Country2").MergeCells = True
End With
```

Review

This tutorial shows how to tell the WebGrid to merge cells on a specific column.

Cell Formatting

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

The WebGrid uses the standard framework format method when formatting data into a column, thus you can use format strings such as *c* for currency and *d* for date, but be aware that if your grid is editable by the user that they may not understand these built-in formats. One way to avoid confusion is to use the long format.

If the built-in format property of the cell does not meet the needs of the application, take a look at [Column Templates](#). This allows you to place any web control in the cell, which provides almost unlimited formatting and editing capabilities.

Questions

- How do I set the format of a cell to display currency and date values?

Solution

Use the **Format** property of the column to tell WebGrid how to format cell values.

Sample Project

This project retrieves a joined dataset from the Northwind database in an SQL Server and displays the data with the currency and date columns formatted.

Perform the following steps to create this sample project.

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known SQL Server data connection or click New Connection and complete the Data Link Properties dialog to create a connection to a Northwind database on SQL Server.
Note This sample project will not run using an Access database. The Access data engine cannot handle the SQL command used.
 - Choose a Query Type: User SQL statements
 - Generate the SQL Statements: Click Query Builder, Add Orders and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. In Solution Explorer, Double-click DataSet1.xsd and select the XML tab. Change the XML to read as follows:

Note The required changes are to remove the `key constraint` section and change the `xs:element` section. The remainder should be the same. You may wish to cut and paste the XML code below.


```

<?xml version="1.0" standalone="yes" ?>
<xs:schema id="DataSet1" targetNamespace="http://www.tempuri.org/DataSet1.xsd" xmlns:
mstns="http://www.tempuri.org/DataSet1.xsd" xmlns="http://www.tempuri.org/DataSet1.
xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-
com:xml-msdata" attributeFormDefault="qualified" elementFormDefault="qualified">
<xs:element name="DataSet1" msdata:IsDataSet="true">
<xs:complexType>
<xs:choice maxOccurs="unbounded">
<xs:element name="Orders">
<xs:complexType>
<xs:sequence>
<xs:element name="CustomerID" type="xs:string" minOccurs="0" />
<xs:element name="OrderID" type="xs:int" minOccurs="0" />
<xs:element name="OrderDate" type="xs:dateTime" minOccurs="0" />
<xs:element name="Amount" type="xs:decimal" minOccurs="0" />
<xs:element name="CompanyName" type="xs:string" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>

```

5. From the menu select Build, Rebuild Solution go get Visual Studio synchronized after making changes to DataSet1.xsd.
6. In Solution Explorer, double-click WebForm1.aspx and the designer should display.
7. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
8. In the Solution Explorer expand the References node. Look for a reference to [Infragistics.WebUI.Shared](#). If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
9. Click the Infragistics.WebUI.Shared reference and set the following properties:
CopyLocal = True
10. Click on the WebGrid and set the following properties:
DataSource = DataSet11
DataMember = Customers
11. Notice the column names in the grid reflect the columns in DataSet1.xsd.
12. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```

If Me.IsPostBack = False Then
    ' build SQL statement to retrieve dataset
    Dim sbSQL As New System.Text.StringBuilder()
    sbSQL.Append(" SELECT o.CustomerID, c.CompanyName, o.OrderID, o.OrderDate,")
    sbSQL.Append(" cast(sum(d.UnitPrice * d.quantity * (1.0 - d.Discount)))")
    sbSQL.Append(" as decimal (10,2)) as Amount FROM Orders as o")
    sbSQL.Append(vbCrLf)
    sbSQL.Append(" right outer join [Order Details] as d on o.OrderID = d.OrderID")
    sbSQL.Append(vbCrLf)
    sbSQL.Append(" right outer join Customers as c on o.CustomerID = c.CustomerID")
    sbSQL.Append(vbCrLf)

```

```

sbSQL.Append(" WHERE o.CustomerID IS NOT NULL")
sbSQL.Append(vbCrLf)
sbSQL.Append(" GROUP BY o.CustomerID, c.CompanyName, o.OrderID, o.OrderDate")

' apply the new sql statement to the select command
Me.OleDbSelectCommand1.CommandText = sbSQL.ToString

' fill the dataset and bind it to the grid
Me.OleDbDataAdapter1.Fill(DataSet1)
Me.UltraWebGrid1.DataBind()

```

End If

- Run the project and, depending on which version of the Northwind database you connect to, you should see something like:

CustomerID	OrderID	OrderDate	Amount	Company
VINET	10248	7/4/1996 12:00:00 AM	440	Vins et alcools CH
TOMSP	10249	7/5/1996 12:00:00 AM	1863.4	Toms Spezialität
HANAR	10250	7/8/1996 12:00:00 AM	1552.6	Hanari Carnes
VICTE	10251	7/8/1996 12:00:00 AM	654.06	Victuailles en stoc
SUPRD	10252	7/9/1996 12:00:00 AM	3597.9	Suprêmes délices
HANAR	10253	7/10/1996 12:00:00 AM	1444.8	Hanari Carnes
CHOPS	10254	7/11/1996 12:00:00 AM	556.62	Chop-suey Chine
RICSU	10255	7/12/1996 12:00:00 AM	2490.5	Richter Supermar

- Observe in the above screen capture the OrderDate shows both the date and time and would be friendlier if it just displayed the date. Also, the amount column would be much more usable if it was formatted as a currency column and right aligned.
- To make these changes, Click on the WebGrid on the WebForm1.aspx designer and:
 - From the Properties dialog select the Columns Collection Editor.
 - Click on the OrderDate column and set the **Format** property to date or (MM/dd/yyyy). You can use any of the built-in format codes like *d*, but if you are allowing for edits in the grid you should use the longer form since the client-side handles this format well.
 - Click on the Amount column and set the **Format** property to currency or (\$ ###,###, ##0.00), and the **CellStyle.HorizontalAlign** property to Right.
- Run the project and you should see something like:

The screenshot shows a Microsoft Internet Explorer browser window titled "WebForm1 - Microsoft Internet Explorer". The address bar displays the URL "/Cell_Formatting/Cell_Formatting.aspx". The main content area contains a table with the following data:

CustomerID	OrderID	OrderDate	Amount	CompanyName
VINET	10248	07/04/1996	\$ 440.00	Vins et alcools Chevalier
TOMSP	10249	07/05/1996	\$ 1,863.40	Toms Spezialitäten
HANAR	10250	07/08/1996	\$ 1,552.60	Hanari Carnes
VICTE	10251	07/08/1996	\$ 654.06	Victuailles en stock
SUPRD	10252	07/09/1996	\$ 3,597.90	Suprêmes délices
HANAR	10253	07/10/1996	\$ 1,444.80	Hanari Carnes
CHOPS	10254	07/11/1996	\$ 556.62	Chop-suey Chinese
RICSU	10255	07/12/1996	\$ 2,490.50	Richter Supermarkt

Review

This project shows how to connect to an SQL Server hosted Northwind database, extract and summarize data from the Orders, Order Details and Customers table, format the date and amount columns and display the table in a WebGrid. Notice the strongly typed dataset which reflects the results of the multiple outer-joined SQL command.

Cell Validation

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Validating user input is a must for most applications and comes in many forms. Minimum data validation requirements usually include:

- Checking to make sure string fields do not exceed a maximum length.
- Checking to make sure number fields are between minimum and maximum values.
- Checking to make sure date fields are valid and between minimum and maximum values.
- Checking to make sure coded fields such as US state codes and international country codes are selectable from a table of valid values.

Advanced user input validation falls into two categories:

- Application of a custom control to a grid column for display and user input. The WebGrid provides this capability through the use of Column Templates.
- Handle character by character input filtering and validation by implementing Client-Side API Cell Validation.

Questions

- How can I perform basic validation of string lengths, number ranges, dates and value lists?

Solution

Use the built-in column validation properties for strings, number ranges and dates.

Use a value list to provide for basic list item selection, and the WebCombo to provide multi-column drop-down list item selection.

Sample Project

This sample project displays selected columns from the Northwind Orders table with the WebGrid updatable property set to true for data entry. The properties of the columns are set to perform basic cell validation.

Perform the following steps to create this project:

Start a new Web Forms project.

1. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements

- Generate the SQL Statements: Click Query Builder, Add Orders and click Close, click (All Columns) and click OK.
 - Click Finish.
2. From the main Menu select Data, Generate Dataset and click OK.
 3. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
 4. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
 5. Click the Infragistics.WebUI.Shared reference and set the following properties:
 - CopyLocal** = True
 6. Click on the WebGrid and set the following properties:
 - DataSource** = DataSet11
 - DataMember** = Customers
 7. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

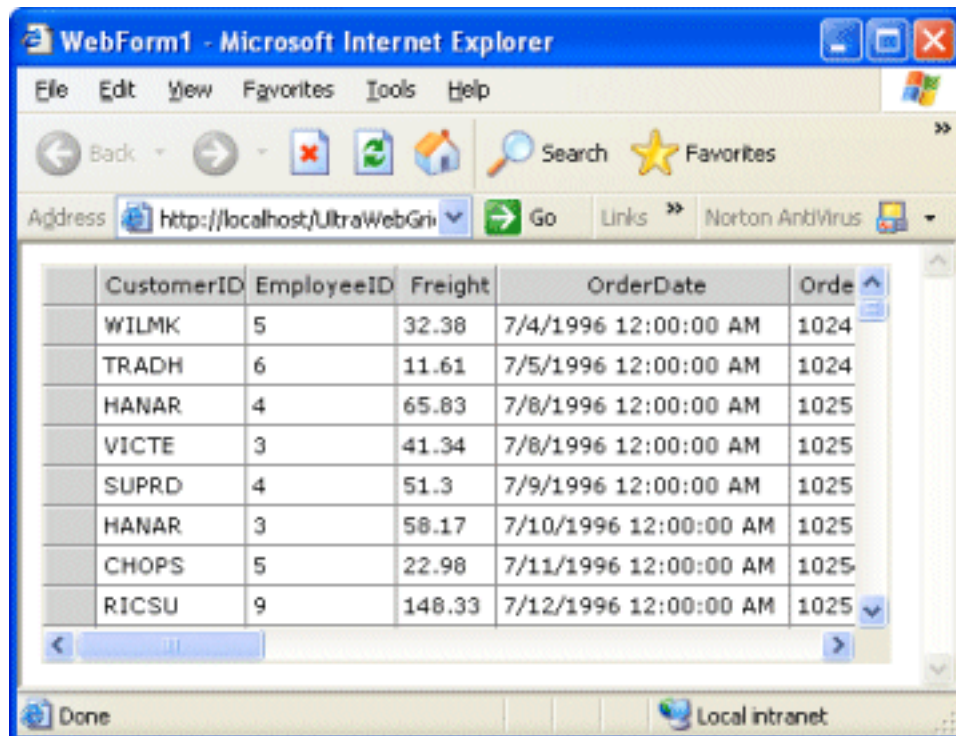
In Visual Basic:

```

If Me.IsPostBack = False Then
    Me.OleDbSelectCommand1.CommandText _
    = "SELECT TOP 50 * FROM Orders"
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If

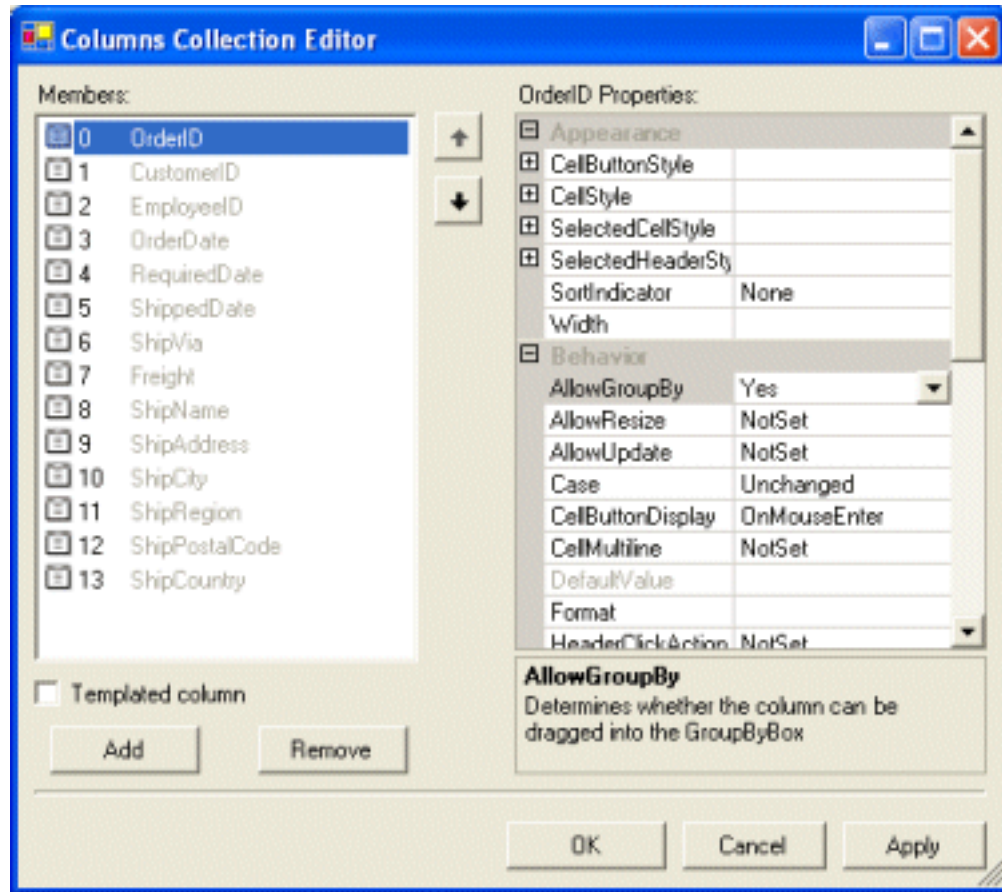
```

8. Run the project and depending on which version of the Northwind database you connect to you should see something like:

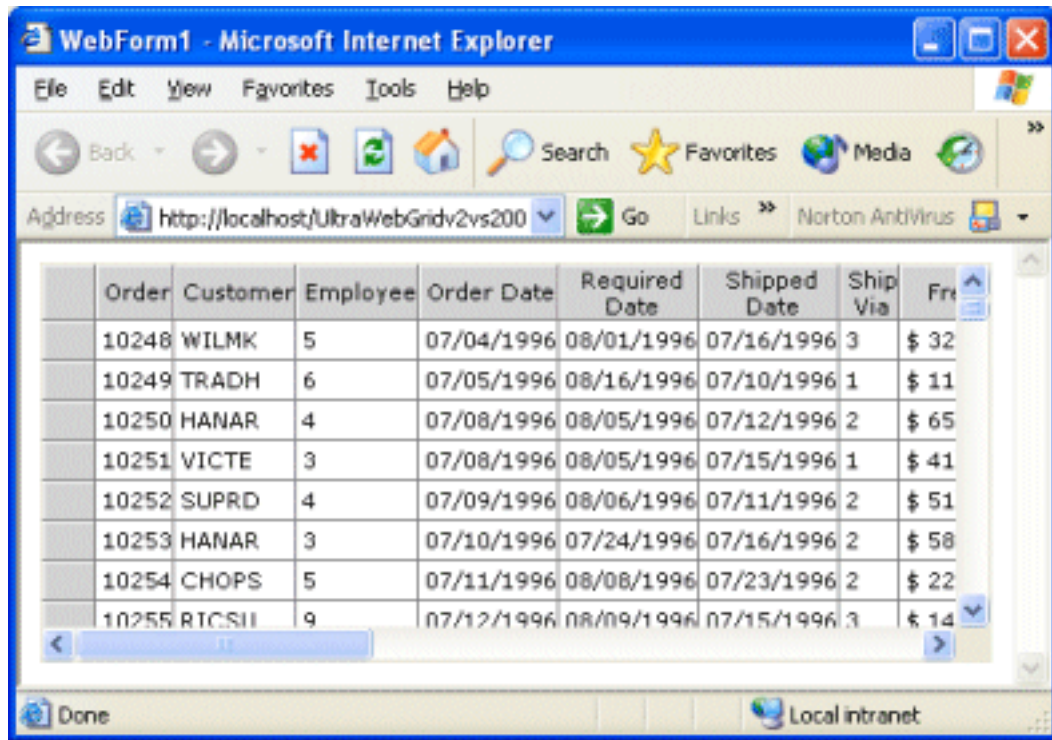


9. Rearrange the column positions into the following order by clicking on the grid in the WebForm1.aspx designer and opening the columns collection editor. You can use the Up/Down arrows rearrange the

columns:



10. Shorten the column header text.
11. Format date and currency column values.
12. Set the FieldLen property for all string fields.
13. Set the property HeaderStyleDefault.**Wrap** = True so the header text will wrap when possible to reduce column width.
14. Run the project and you should see something like:



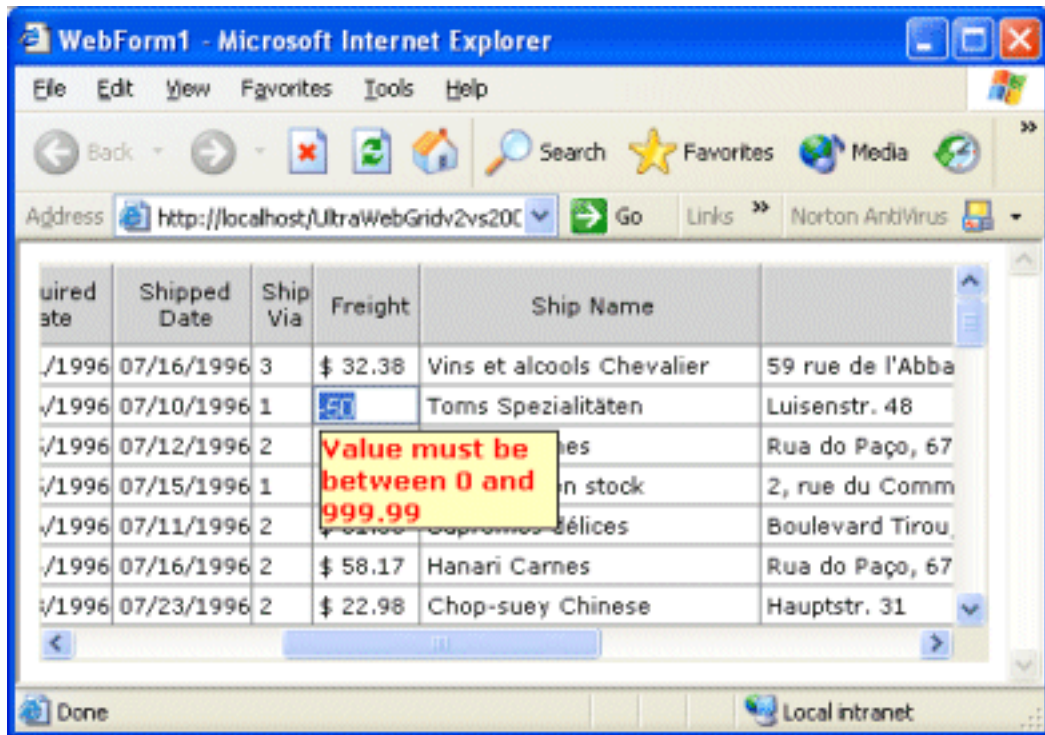
- After the appearance changes made in the previous steps, set the following property to make the grid updateable and a single click will invoke cell edit:

DisplayLayout.**AllowUpdateDefault** = Yes
 DisplayLayout.**CellClickActionDefault** = Edit

- Run the project and see if you can enter a date incorrectly. Notice that any valid date is accepted, but any invalid date is simply rejected.
- Try entering more characters than are specified in the **FieldLen** property of a string cell. Notice that you can not enter more characters than specified.
- See what happens when you try to enter a currency value in the Freight column. Try entering a negative value in the Freight column. Notice it will take about anything you want to enter. Terminate the program.
- Add a validator to only allow for positive values in the Freight column. From the toolbox, drag and drop a RangeValidator on the designer below the WebGrid. Click on this validator control and set the following properties:

BackColor = #FFFFFF
BorderColor = Black
BorderStyle = Solid
BorderWidth = 1px
ControlToValidate = UltraWebGrid1
ErrorMessage = Value must be between 0 and 999.99
Font = Verdana, X-Small
MaximumValue = 999.99
MinimumValue = 0

- Click on the WebGrid and in the Columns collection select the Freight column. Open the Validators Collection Editor and select RangeValidator1 and click > to move this validator to the Applied validators: list. Click OK, and click OK to exit the Columns Collection Editor.
- Run the project and observe the behavior of data entry in the Freight column. Notice that if you try to enter a value of -50, the error message displays and you cannot exit the cell without entering a valid value.



Terminate the program when you are finished.

22. To this point almost all of your grid settings have been made using the designers. To apply a value list on the Employee column with the value list entries being retrieved from the Northwind Employees table perform the following steps:
 - a. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - b. On Choose Your Data Connection: select the same connection you used earlier.
 - c. Choose a Query Type: User SQL statements
 - d. Generate the SQL Statements: Click Query Builder, Add Employees and click Close, click (All Columns) and click OK.
 - e. Click Finish.
 - f. From the main Menu select Data, Generate Dataset. Select the New radio button. In the Choose tables to add to dataset, choose only Employees. Click OK.
 - g. Add the following code to the WebGrid **InitializeLayout** event:

In Visual Basic:

```

' load employees and create value list
Me.OleDbDataAdapter2.Fill(DataSet21, "Employees")

' declare and create a new value list
Dim vlEmployees As New Infragistics.WebUI.UltraWebGrid.ValueList()

' tell the column to be a drop down list and bind the value list
With e.Layout.Bands(0).Columns.FromKey("EmployeeID")
    .Type = Infragistics.WebUI.UltraWebGrid.ColumnType.DropDownList
    .ValueList = vlEmployees
End With

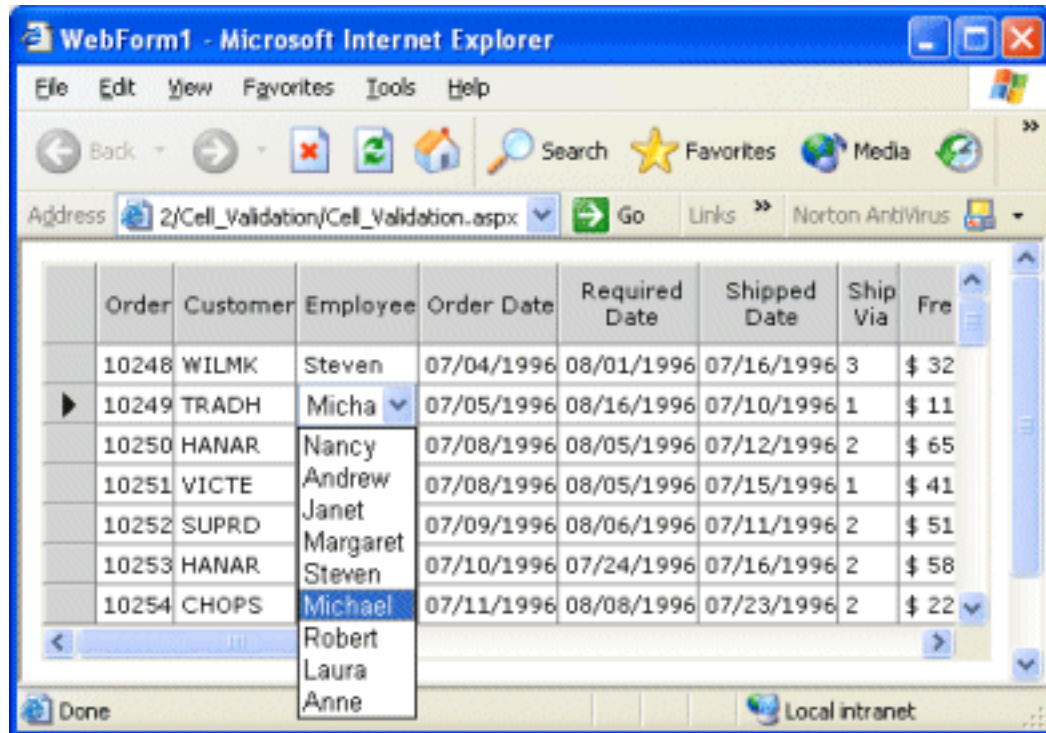
' set value list properties and bind to data
vlEmployees.ValueMember = "EmployeeID"

```



```
vlEmployees.DisplayMember = "FirstName"  
vlEmployees.DataSource = DataSet21  
vlEmployees.DataMember = "Employees"  
vlEmployees.DataBind()
```

- Run the program, observe that the employee name now displays instead of the employee number. Click on an employee cell and notice the drop down list:



Review

This tutorial shows how to perform basic user input validation of string length, number range and coded columns.

WebCombo In Grid Cell

There are 6 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Code Discussion](#)
- [Review](#)

Background

Editing of coded columns should allow the user to request a drop-down-list and select an entry. The UltraWebGrid allows the developer to place a UltraWebCombo control over a grid cell and use the same control as the value list. This allows a code column to display with descriptive text and allows the user to select entries from a drop-down-list.

Questions

- How do I display the CompanyName for a column containing a CustomerID ?
- How do I allow the user to select the CompanyName from a drop-down-list for a column containing a CustomerID?

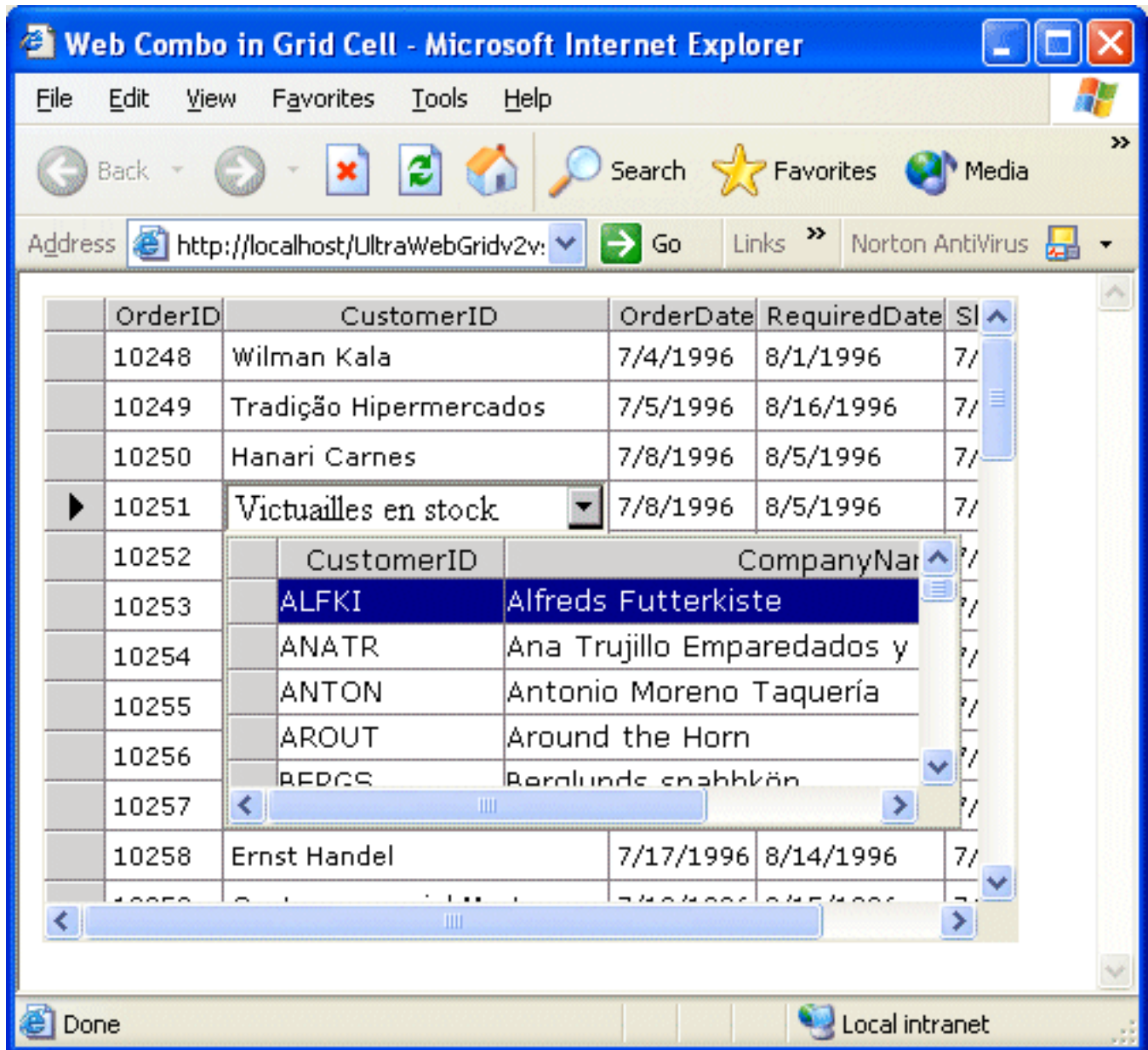
Solution

If the grid or column is display only (no editing), the CompanyName can be displayed in a column containing a CustomerID by using a ValueList. See tutorial *Value Lists*. If the column is editable, The CompanyName can be displayed in a column containing a CustomerID by setting the value list to a web combo control.

To provide a multi-column drop-down-list for a column, set the value list to a WebCombo control.

Sample Project

This sample project retrieves the Northwind Orders database and binds an UltraWebCombo control to the CustomerID column to display the CompanyName. The UltraWebCombo control is populated with values from the Northwind Customers table and displays as a multi-column drop-down when the user selects the cell for editing.



Note This tutorial requires the `uwg2_Helper.vb` module. You can obtain this file by right-clicking on [this link](#) and choosing "Save Target As..." from the context menu. When the File Save dialog appears, save the file in the same folder as the other files that make up the project. You should then add this file to your project.

Code Discussion

Files containing code in this project:

`uwg2_Helper.vb`

This code is not reviewed in this exercise.

`Web_Combo_In_Grid_Cell.aspx`

`Web_Combo_In_Grid_Cell.aspx` contains the code relevant to this project and consists of the following code regions:

Page Events

The Page Events Region contains the following event handlers:

- **Page_Load** - The `Page_Load` event contains code to retrieve the Northwind Orders and Customers and bind them to the grid and combo.

First, declare a new data connection and retrieves the connection string:

In Visual Basic:

```
' create a data connection
Dim cnNorthwind As New OleDb.OleDbConnection()
cnNorthwind.ConnectionString = _
UWG2.Utility.DataConnection.ConnectionString()
```

```
If Me.IsPostBack = False Then
```

Declare a data adapter to retrieve the CustomerID and CustomerName fields from the Customers database. Declare a new data table and fill it with data. Tell the web combo control which column to use as the DataValueField and which column to use as the DataTextField, set the data source to the Customers data table and bind the data to the control.

In Visual Basic:

```
' retrieve and bind customers to web combo
Dim daCustomer As OleDb.OleDbDataAdapter = _
UWG2.Utility.OleDbDataAdapter.CreateSelectOnly( _
"SELECT CustomerID, CompanyName FROM Customers", cnNorthwind)

Dim dtCustomer As New DataTable("Customers")

daCustomer.Fill(dtCustomer)
WebCombo1.DataValueField = "CustomerID"
WebCombo1.DataTextField = "CompanyName"
WebCombo1.DataSource = dtCustomer
WebCombo1.DataBind()
```

Declare a data adapter to retrieve rows from the Orders database. Declare a new data table and fill it with data using the data adapter. Set the data source of the grid to the data table and bind the data to the grid.

In Visual Basic:

```
' retrieve and bind orders to web grid
Dim daOrder As OleDb.OleDbDataAdapter = _

UWG2.Utility.OleDbDataAdapter.CreateSelectOnly( _
"SELECT TOP 50 OrderID, CustomerID, OrderDate" _
+ ", RequiredDate, ShippedDate FROM Orders", cnNorthwind)

Dim dtOrder As New DataTable()

daOrder.Fill(dtOrder)
UltraWebGrid1.DataSource = dtOrder
UltraWebGrid1.DataBind()

End If
```

UltraWebGrid Events

The UltraWebGrid Events Region contains the following event handlers:

- **UltraWebGrid1_InitializeLayout** - The UltraWebGrid **InitializeLayout** event contains code to set the layout properties of the control:

In Visual Basic:

```
With e.Layout.Bands(0)
```

To make the grid editable, set the **CellClickAction** property to Edit and set the **AllowUpdate** property to Yes

In Visual Basic:

```
' turn on click action edit and allow update so  
' the web combo will(show)  
.CellClickAction = _  
Infragistics.WebUI.UltraWebGrid.CellClickAction.Edit  
.AllowUpdate = Infragistics.WebUI.UltraWebGrid.AllowUpdate.Yes
```

To bind the web combo to the CustomerID column, set the **Type** property to DropDownList, set the ValueList.WebCombo control to the WebCombo1 control. To set the column to display the text rather than the value, set the **DisplayStyle** property to DisplayText.

In Visual Basic:

```
' apply web combo to CustomerID column  
.Columns.FromKey("CustomerID").Type _  
= Infragistics.WebUI.UltraWebGrid.ColumnType.Custom  
.Columns.FromKey("CustomerID").EditorControlID = WebCombo1.UniqueID  
.Columns.FromKey("CustomerID").ValueList.DisplayStyle _  
= Infragistics.WebUI.UltraWebGrid.ValueListDisplayStyle.DisplayText
```

Format the date columns to make them look a little better.

In Visual Basic:

```
' format the date columns to make them look better  
.Columns.FromKey("OrderDate").Format = "d"  
.Columns.FromKey("RequiredDate").Format = "d"  
.Columns.FromKey("ShippedDate").Format = "d"  
End With
```

UltraWebCombo Events

The UltraWebCombo Events Region contains the following event handlers:

- **WebCombo1_InitializeLayout** - The web combo control defaults the column widths to about 100 pixels. To display more of the CompanyName column, set the width to about 300 pixels:

In Visual Basic:

```
With e.Layout.Bands(0)  
.Columns.FromKey("CompanyName").Width = New System.Web.UI.WebControls.Unit(300)  
End With
```

Review

This tutorial shows how to bind an UltraWebCombo control to a column for display of text information and multi-column drop-down selection.

Outlook GroupBy

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

The WebGrid ViewType property supports Flat, Hierarchical and Outlook Group By modes. The Outlook Group By view type is a combination of Flat and Hierarchical, and the user gets to decide what hierarchical groupings to use.

Questions

- How do I allow the user to display my Flat data in a hierarchical view format, and the user can select the hierarchical grouping?

Solution

Populate the WebGrid with flat data as normal and set the WebGrid View Type to Outlook GroupBy.

Sample Project

This sample project shows how to connect to the Northwind Customers database and tell the WebGrid how to display the customers in Outlook Group By format.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb
 - Choose a Query Type: User SQL statements
 - Generate the SQL Statements: Click Query Builder, Add Customers and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

6. Click on the WebGrid and set the following properties:

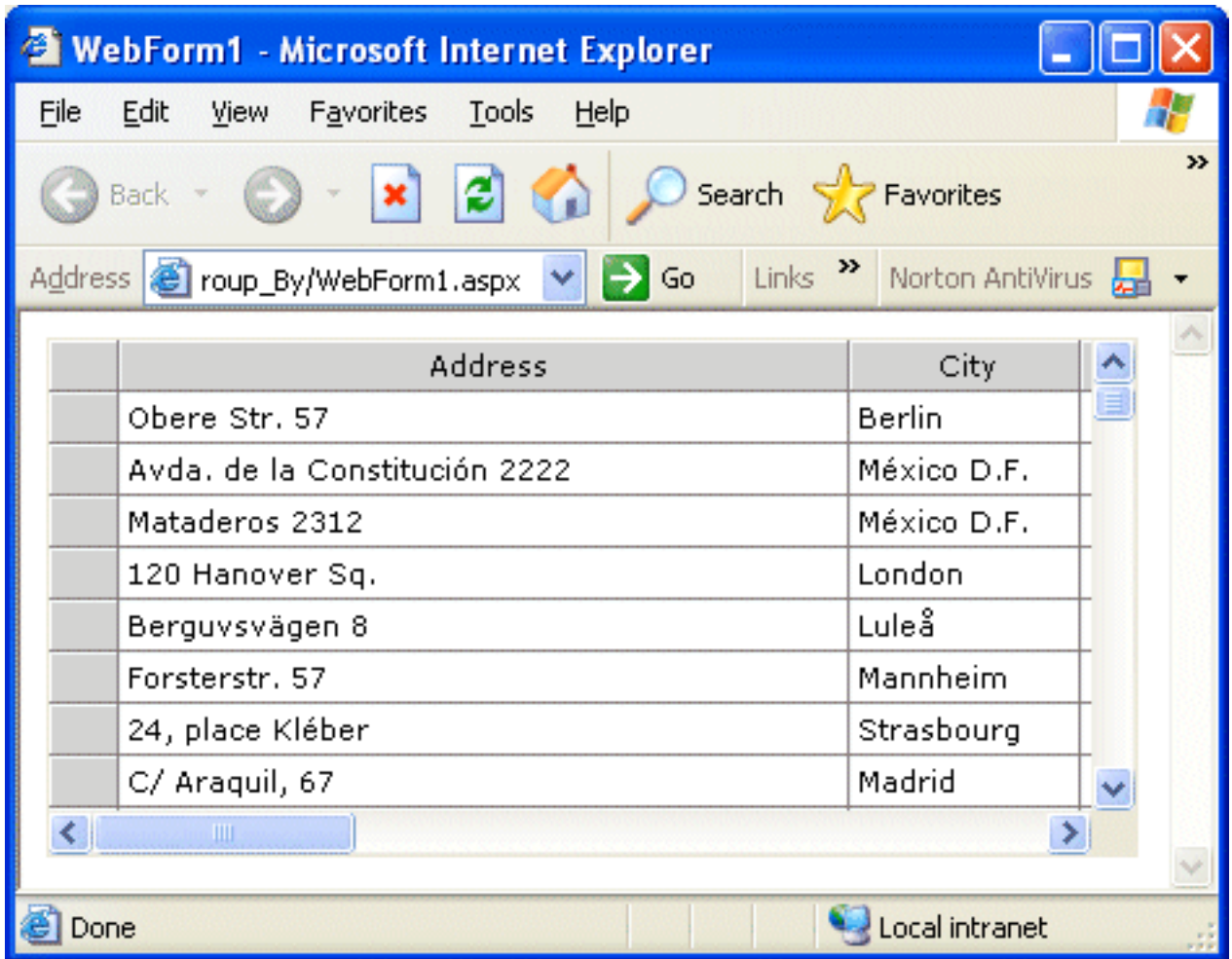
DataSource = DataSet11 **DataMember** = Customers **Width** = 420px

7. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    OleDbDataAdapter1.Fill(DataSet11)
    UltraWebGrid1.DataBind()
End If
```

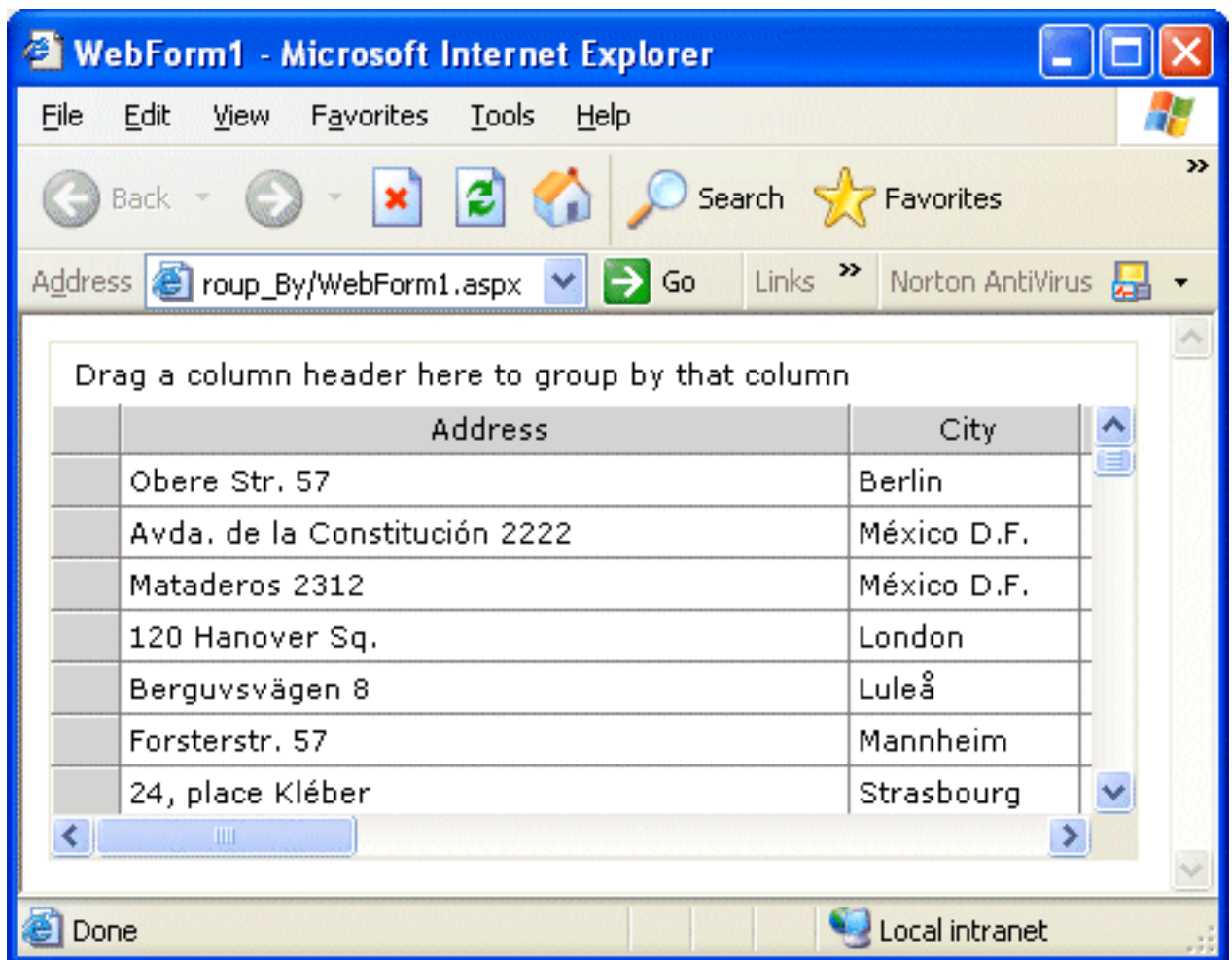
8. Run the project, and depending on the version of the Northwind database you should see something like:



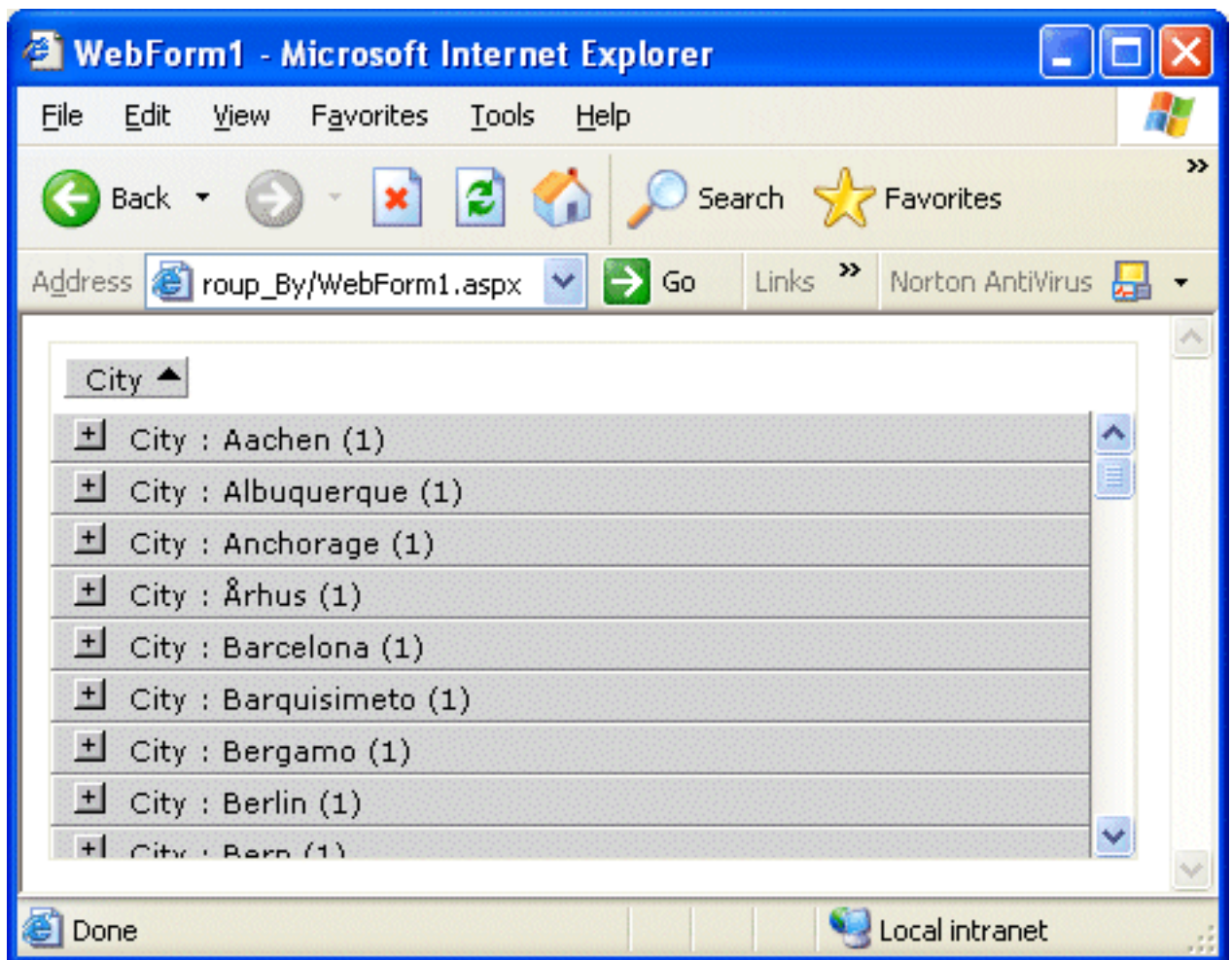
9. To change the grid to display in Outlook Group By mode, click on the WinGrid in the designer and set the following property:

DisplayLayout.**ViewType** = OutlookGroupBy

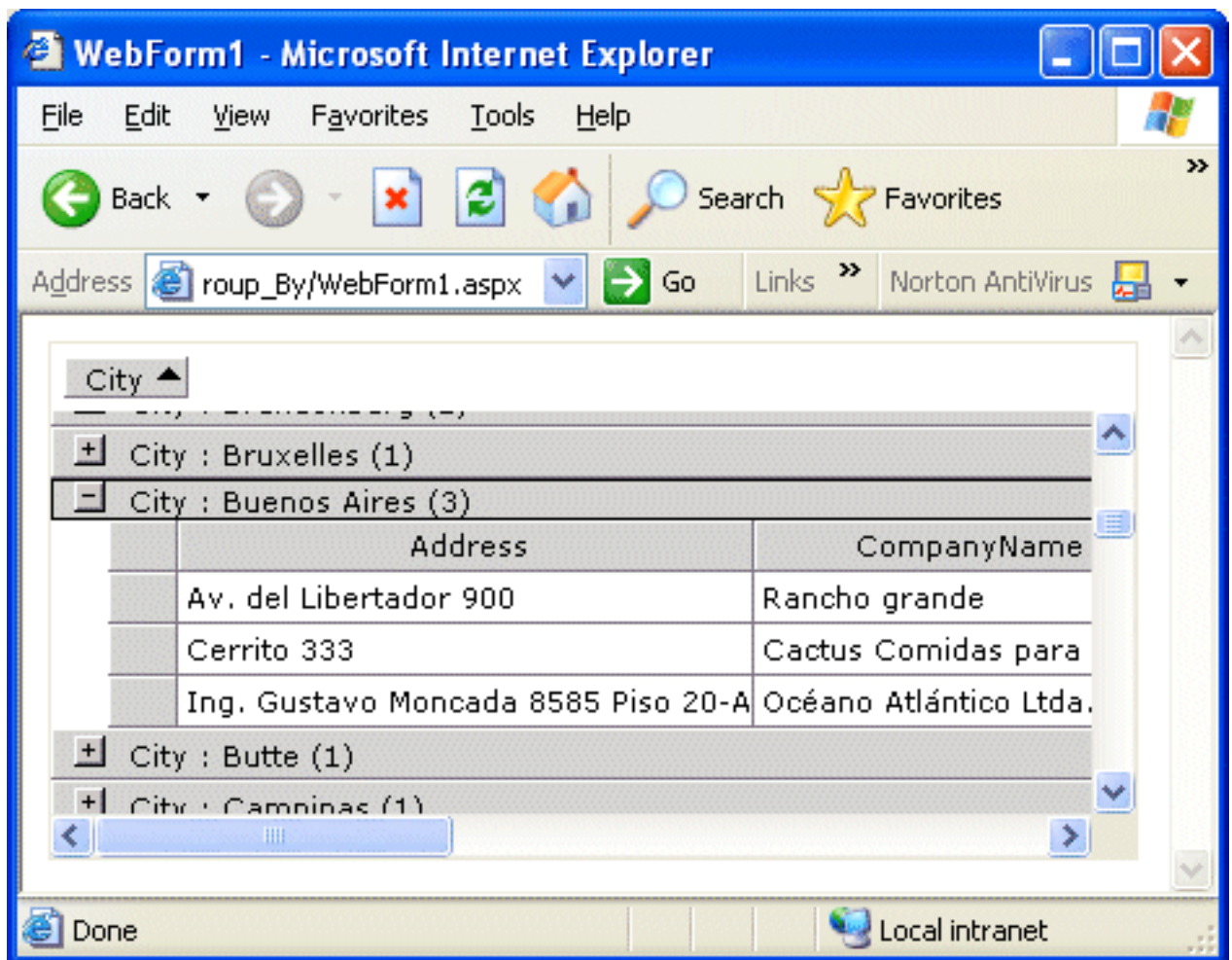
10. Run the project and you should see something like:



11. Drag the City column header to the group by area. The grid will perform a post back to the server and redisplay something like:



12. Now you have a hierarchical style grid listing the cities on the left with a count of the number of records for each city. The following shows a city with three customers:



Review

This tutorial shows how to bind the WebGrid to the Northwind Customers database table and display the results in Outlook Group By format.

Read Only Modes

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Many applications simply display data. For these applications the WebGrid will be ReadOnly and do not require view state and a lot of jScript code to be sent to the browser.

The WebGrid supports two levels of ReadOnly mode:

LevelZero

LevelZero ReadOnly mode sends no jScript to the browser and thus only supports scrolling. Use this mode if you are displaying flat data and only need scrolling.

LevelOne

LevelOne ReadOnly mode sends a minimum of jScript to the browser to support scrolling, row expansion and collapse, and row and column resizing. Use this mode if you are displaying hierarchical data.

Questions

- All I want to do is display data in the grid. Some contain flat data and others are hierarchical. How can I reduce the amount of text being sent to the browser to a minimum?

Solution

- Use ReadOnly mode LevelZero if you are displaying flat data.
- Use ReadOnly mode LevelOne if you are displaying hierarchical data.

Sample Project

This project first displays Northwind Customer data in a flat format using ReadOnly mode Level Zero, then the project is expanded to display Northwind Customers and Orders data in a hierarchical format using ReadOnly mode Level One.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.

- Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
 4. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
 5. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True
 6. Click on the WebGrid and set the following properties:

DataSource = DataSet11
DataMember = Customers

DisplayLayout.ReadOnly = LevelZero
EnableViewState = False
 7. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

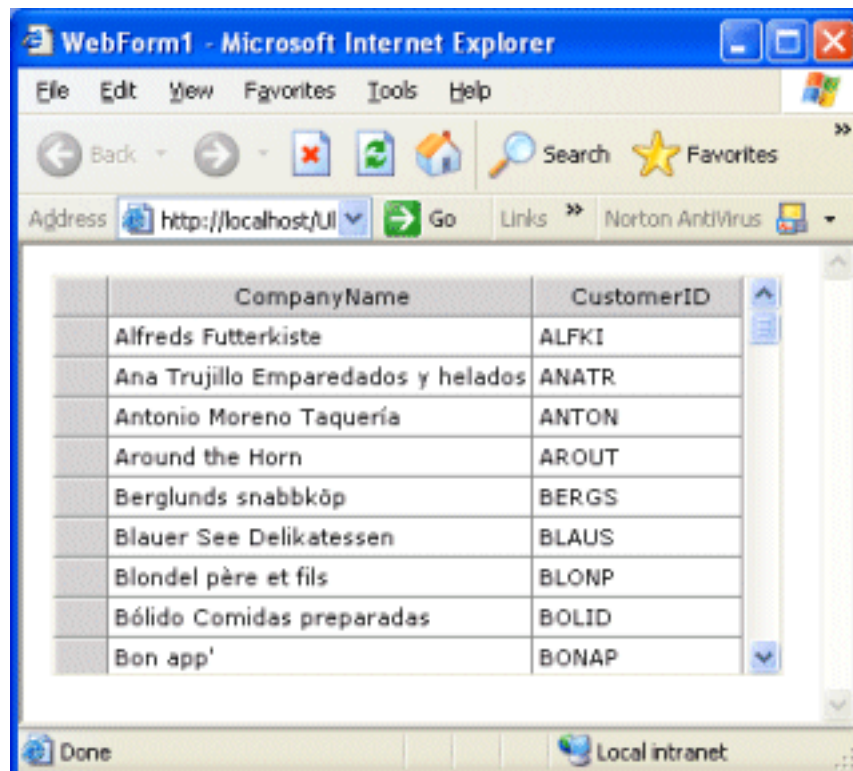
In Visual Basic:

```

If Me.IsPostBack = False Then
    OleDbDataAdapter1.Fill(DataSet11)
    UltraWebGrid1.DataBind()
End If

```

8. Run the project and depending on which version of the Northwind database you connect to you should see something like:



9. In the browser select View, Source and observe that there is very little view state information and very little JavaScript.

10. To add the Orders table to this project: From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - Choose the same data connection you used for the Customers adapter.
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Orders and click Close, select CustomerID, OrderID, ShipCity, ShipCountry and click OK.
 - Click Finish.
11. From the main Menu select Data, Generate Dataset and click OK.
12. In Solution Explorer double-click DataSet1.xsd to open the data set designer. To add a new relationship between the tables on CustomerID perform the following steps:
 - On the Customers table, right-click the CustomerID column and select Add, New Relation.
 - In the Edit Relation dialog, change the Child element to Orders and click OK.
13. You have updated the data set, but the WebGrid does not yet know about your changes. To configure the WebGrid with your new data set changes and LevelOne ReadOnly mode, perform the following steps:
 - In the Solution Explorer double-click the WebForm1.aspx node to open the designer.
 - Click on the WebGrid and set the properties in the following order (The order of these settings is important because you are clearing and resetting some of them to have the grid reflect the changes made to the data set.)

DataSource = *(remove this entry)*

DataMember = *(remove this entry)*

DataSource = DataSet11

DataMember = Customers

DisplayLayout.ReadOnly = LevelOne

DisplayLayout.ViewType = Hierarchical

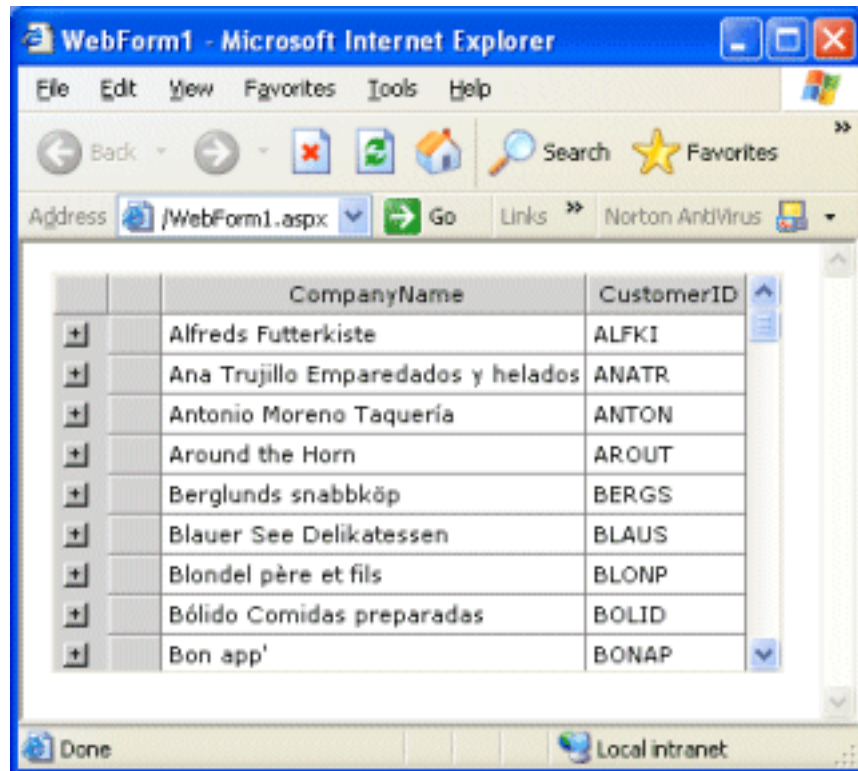
EnableViewState = False *(should already be set)*

14. Add the following line of code to the Page Load event to fill the Orders data adapter before the data bind:

In Visual Basic:

```
OleDbDataAdapter2.Fill(DataSet11)
```

15. Run the project. It will still take a while to create and transmit this file because there are many orders for each customer. This would be an ideal place to implement some paging. You should see something like:



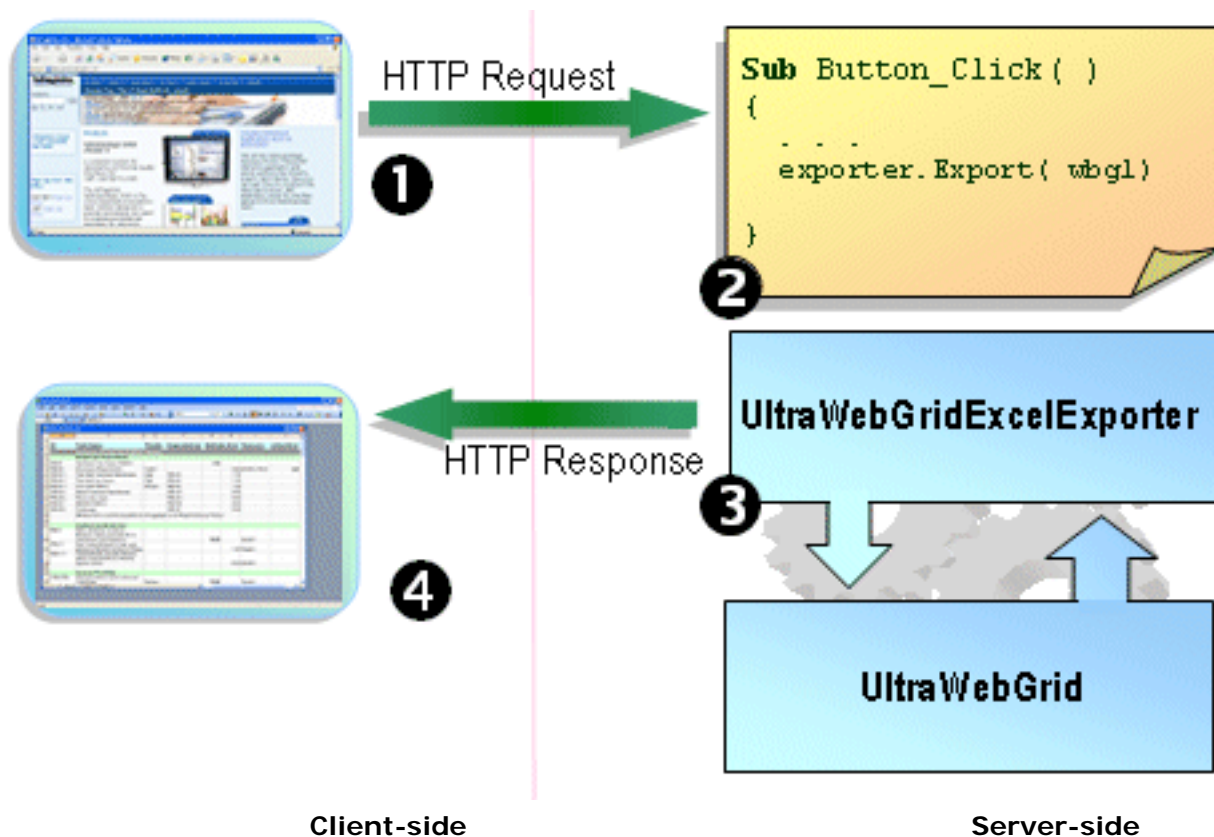
16. In the browser view the source and you will see there is very little JavaScript or view state information. Expand and Collapse some of the customer nodes and observe the order records.

Review

This tutorial describes and illustrates the use of the WebGrid read only level zero and level one mode. These read only mode help increase response time by reducing network traffic when flat or hierarchical data is displayed for user viewing.

Exporting Grid Data To Excel

With this release, the UltraWebGrid has the ability to export data in Microsoft Excel's native spreadsheet format. Exporting to Excel format is a process similar to rendering grid data, in that developers have control over how Layouts and Appearances are applied to the data before export, and which data should be included or excluded.



The figure above illustrates the Export to Excel processing in its most common scenario, as described in the overview entitled [Exporting Grid Data to Excel Format](#). The following procedures describe how developers may customize the export processing at each step.

1. The end user triggers a user-interface device (for instance, by clicking on a Button labeled "Export") that posts back an HTTP Request to the server hosting the web application.
2. The developer's event handler for this action that is supposed to trigger Export to Excel processing performs any necessary preliminary processing, and calls the **Export** method on the **UltraWebGridExcelExporter** supplying a reference to the **UltraWebGrid** containing the grid data intended for download to the end user.
3. **UltraWebGridExcelExporter** walks the object model of the **UltraWebGrid** it is given at run-time to convert the grid's tabular data and appearances into formatted Excel cells. At various points throughout this translation, **UltraWebGridExcelExporter** fires events that developers can handle to customize how the translation is performed.
4. All other web controls appearing on the web form have their rendering suppressed such that the content formulated in an HTTP Response to the end user's browser represents only binary data in the native Excel spreadsheet file format. Developers may customize whether the Excel application should be in-place activated within the end user's browser, or launched as a separate application, by default.

Based on the configuration of the end user's workstation, they may receive a dialog box warning them about downloading the spreadsheet file. It may offer the end user the ultimate choice in how the document should be opened, or saved to their local file system. This dialog, when present, is a security feature of end users' workstations and cannot be overridden by the control.

To export grid data to Excel format, take the following steps:

1. Add an **UltraWebGrid** to a form in your application. (Or open an existing application and display a web form that contains the UltraWebGrid. Make sure your application is using V3 or later of the UltraWebGrid control.)
2. Locate the **UltraWebGridExcelExporter** control in the Visual Studio Toolbox and double-click it to add it to the form. It appears as a placeholder in the form's designer, but does not render this placeholder at run-time. Other web controls and text may be positioned over it. (If you cannot locate the control in the Toolbox, right-click to display the context menu for the Toolbox, select "Customize Toolbox..." and then click the .NET Framework tab if it is not already selected. Scroll down until you see the UltraWebGridExcelExporter listed, then check its checkbox and click OK.)
3. Add a web control to the form (such as a command button) that will be responsible for initiating the export function at the appropriate time within the web application.
4. In the **Click** (or equivalent) event of the export-initiating control, enter the following line of code:

In Visual Basic:

```
Me.UltraGridExcelExporter1.Export( Me.UltraGrid1)
```

In C#:

```
this.ultraGridExcelExporter1.Export( this.ultraGrid1);
```

This code begins the export process by invoking the **Export** method of the **UltraWebGridExcelExporter** control. When invoking this method, it is necessary to pass the **UltraWebGrid** that is the source of the exported data. There are a number of overloads for this method, each allowing greater customization over the export process.

If the sole requirement is to dump all of the grid data into an Excel file, retaining any current formatting, this is all a developer must do.

5. Many of the customizations possible through overloads of the **Export** method can also be accomplished through the Properties Editor window at design-time, or set programmatically. Here is some example code that an application might use to offer the end user text boxes permitting them to customize the filename of the exported spreadsheet, and the name of the Excel Worksheet on which the grid data will appear.

In Visual Basic:

```
Public Sub TextBox1_TextChanged( ByVal sender As Object, ByVal e As System.EventArgs)

    ' Set the spreadsheet filename in the download to user-specified filename
    ' if it is not the empty string.
    If ( Me.TextBox1.Text.Length > 0 ) Then
        Me.UltraGridExcelExporter1.DocumentName = Me.TextBox1.Text
    End If

End Sub

Public Sub TextBox2_TextChanged( ByVal sender As Object, ByVal e As System.EventArgs)

    ' Set the worksheet tab name inside of the downloaded spreadsheet if it is
    ' not the empty string.
    If ( Me.TextBox1.Text.Length > 0 ) Then
        Me.UltraGridExcelExporter1.WorksheetName = Me.TextBox2.Text
    End If

End Sub
```


In C#:

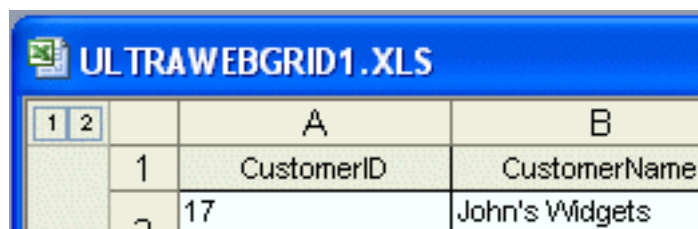
```
private void TextBox1_TextChanged( object sender, System.EventArgs e)
{
    // Set the spreadsheet filename in the download to user-specified filename
    // if it is not the empty string.
    if ( this.TextBox1.Text.Length > 0 )
    {
        this.UltraGridExcelExporter1.DocumentName = this.TextBox1.Text;
    }
}

private void TextBox2_TextChanged( object sender, System.EventArgs e)
{
    // Set the worksheet tab name inside of the downloaded spreadsheet if it is
    // not the empty string.
    if ( this.TextBox2.Text.Length > 0 )
    {
        this.UltraGridExcelExporter1.WorksheetName = this.TextBox2.Text;
    }
}
```

The **WorksheetName** and **DownloadName** properties are very strict regarding the input they accept for security reasons. These names must not exceed 32 characters, and can only contain alphanumeric characters and the period commonly used to denote a file system extension. Any sort of absolute or relative path is prohibited.



WorksheetName



	A	B
1	CustomerID	CustomerName
2	17	John's Widgets

DownloadName

When no value is specified for these properties, the **WorksheetName** defaults to "Sheet1" and the **DownloadName** will be based upon the programmatic identifier of the exported **UltraWebGrid** (for example, if the control has an **ID** of "UltraWebGrid1" then the default filename for the downloaded spreadsheet will be "UltraWebGrid1.xls").

There are additional properties supplied for customizing the starting row and column at which the grid data appears in an exported Excel spreadsheet. Developers may use the **ExcelStartRow** to create space for a corporate letterhead above the exported grid data, or the **ExcelStartColumn** to create a left margin that can conceal derived columns or hidden formulae. These properties parallel another overload of the **Export** method which takes the *startingRow* and *startingColumn* specified in a zero-based index (instead of an Excel-friendly, one-based identifier as supported by the properties).

6. If it is necessary to fine-tune the exported data, developers may do so by using the events fired by the **UltraWebGridExcelExporter** control. Working with the **CellExporting** event (or any of the other ...Exporting events) grants access to individual cells of grid data before it is written into the Excel file, so developers can perform pre-processing. Post-processing can also be performed on the data after it has been written into the file by using the **CellExported** event. For example, the following code renders dates prior to today's date in red (for example, overdue activities in grid data describing project deadlines):

In Visual Basic:

```
Public Sub UltraWebGridExcelExporter1_CellExporting( ByVal sender As Object, _
    ByVal e As Infragistics.WebUI.UltraWebGrid.ExcelExport.CellExportingEventArgs)
```

```

    If ( ( e.Value Is Not Nothing ) And ( CDate( e.Value ) < System.DateTime.Now ) )
Then
    Dim cfCellFmt As Infragistics.Excel.IWorksheetCellFormat
    Dim iRdex As Integer = e.CurrentRowIndex
    Dim iCdex As Integer = e.CurrentColumnIndex

    ' Obtain reference to CellFormat object for current cell.
    cfCellFmt = e.CurrentWorksheet.Rows( iRdex).Cells( iCdex).CellFormat

    ' Set format property for the font color to red.
    cfCellFmt.Font.Color = System.Drawing.Color.Red

    ' Apply the formatting, this step minimizes the number of
    ' Worksheet Font objects that need to be instantiated.
    e.CurrentWorksheet.Rows( iRdex).Cells( iCdex).SetFormatting( cfCellFmt)
End If

End Sub

```

In C#:

```

public void UltraWebGridExcelExporter1_CellExporting( object sender, _
    Infragistics.WebUI.UltraWebGrid.ExcelExport.CellExportingEventArgs e)
{
    if ( ( e.Value != null ) && ((DateTime)( e.Value)) < System.DateTime.Now )
    {
        Infragistics.Excel.IWorksheetCellFormat cfCellFmt;
        int iRdex = e.CurrentRowIndex;
        int iCdex = e.CurrentColumnIndex;

        // Obtain reference to CellFormat object for current cell.
        cfCellFmt = e.CurrentWorksheet.Rows[ iRdex].Cells[ iCdex].CellFormat;

        // Set format property for the font color to red.
        cfCellFmt.Font.Color = System.Drawing.Color.Red;

        // Apply the formatting, this step minimizes the number of
        // Worksheet Font objects that need to be instantiated.
        e.CurrentWorksheet.Rows[ iRdex].Cells[ iCdex].SetFormatting( cfCellFmt);
    }
}

```

Witness the technique demonstrated by this code for obtaining the **WorksheetCell** based on the **CurrentRowIndex** and **CurrentColumnIndex** provided by the event arguments. This approach will be applicable in many scenarios working with Export to Excel events. Each **WorksheetCell** goes on to expose an implementation of the *IWorksheetCellFormat* interface which allows Excel-specific appearance, layout and style attributes to be assigned.

The call to the **SetFormatting** method is important because cell formats consume space and resources in the native Excel file format, which is limited in the number of cell formats, color palette entries (maximum 56), and fonts which it can represent. This method call checks the cache, and uses a cached format, color or font when one already exists. If the export process appears to be consuming too much time, verify that the number of formats, colors and fonts are kept reasonably small and that **SetFormatting** has been used regularly to collapse duplicate cell formats.

Save and Load XML Layouts

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Visual Studio provides a WYSIWYG designer for web forms, but each web form is ultimately rendered as HTML. The HTML of the web form contains all of the information about the form's design that you have set up at design-time. Since the underlying HTML is where all of your designer-based property settings reside, it is a good idea to periodically save your WebGrid layout. You can use this saved layout to recover all of the property settings in the event you delete the WebGrid from the form designer and replace it with a new entry from the Toolbox, or if the HTML of the form is somehow corrupted and you must re-design the form from scratch.

You can also use saved layouts to duplicate all of your property settings from one grid to another, or you can have multiple layouts with different appearance settings and load the different layouts based on user preferences. There are many reasons for persisting and restoring the grid's layout. The WebGrid saves and loads layouts to XML where it can be easily viewed and even modified if you are careful.

Questions

- How can I save my WebGrid layout so I can load it into a different grid on another form?
- How can I save multiple layouts with different appearance settings and load them at run-time based on user preferences?

Solution

Use the designer options to save and load WebGrid layouts. You can right-click the WebGrid on the form designer and click either Load Layout or Save Layout.

Sample Project

This sample project uses the Northwind Customers data table to populate the WebGrid.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.

4. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
5. Click the Infragistics.WebUI.Shared reference and set the following properties:
CopyLocal = True
6. Click on the WebGrid and set the following properties:
DataSource = DataSet11
DataMember = Customers
7. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

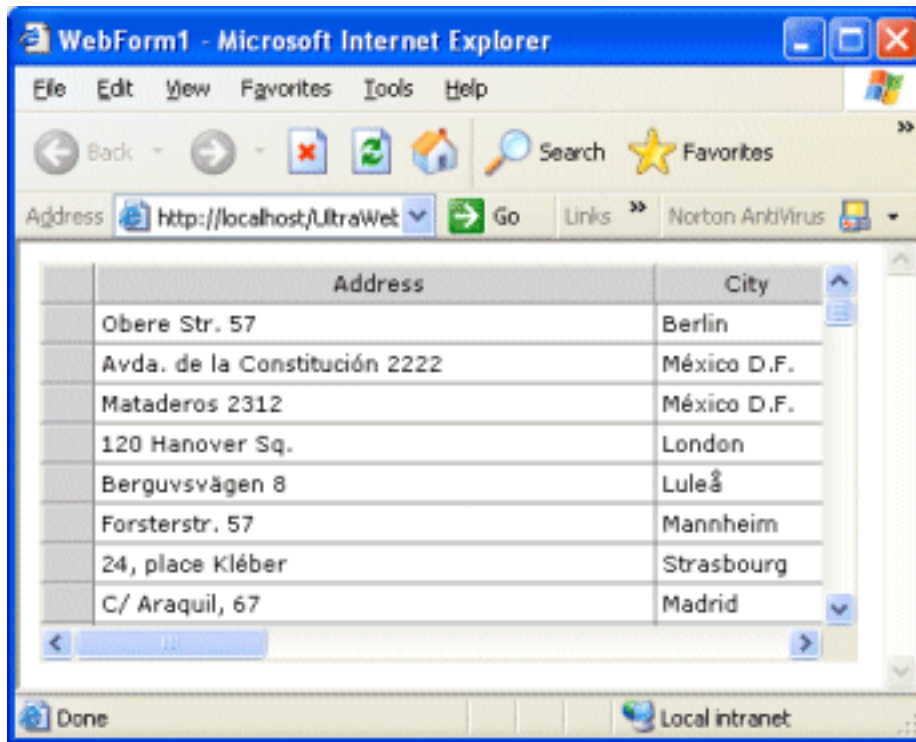
In Visual Basic:

```

If Me.IsPostBack = False Then
    Me.OleDbSelectCommand1.CommandText _
    = "SELECT TOP 100 * FROM [Order Details] ORDER BY Discount DESC"
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If

```

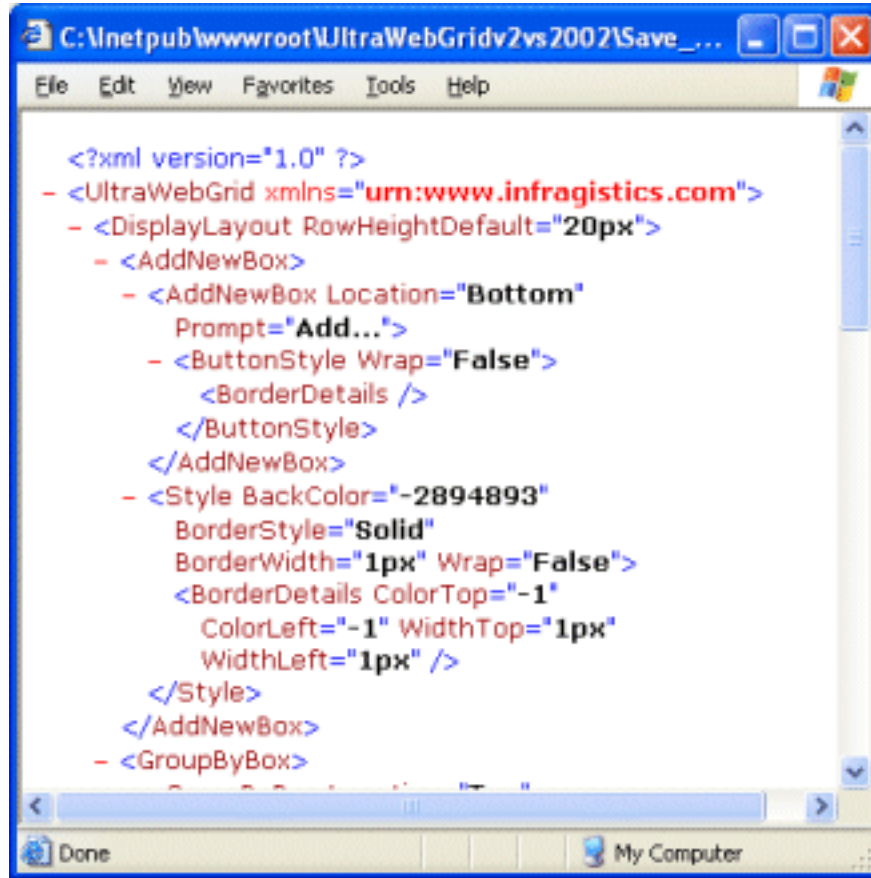
8. Run the project and depending on which version of the Northwind database you connect to you should see something like:



9. Perform the following steps to arrange the columns, set some appearance properties and save the layout as DefaultLayout.xml:
 - On the form designer, right-click the WebGrid and select Edit Layout and set the following properties:
HeaderStyleDefault.BackColor = ActiveCaption
HeaderStyleDefault.ForeColor = InactiveCaptionText
RowStyleDefault.BackColor = InactiveCaptionText

- Click OK

10. Save the layout: On the form designer, right-click the WebGrid and select Save Layout. Save the layout in the same directory where your project resides with the File name of DefaultLayout.xml. To review review the XML , locating the default layout XML file and double-click it and the text should display in the browser something like:



11. If you review this XML you will see that there are many properties other than those you set. These are apparently the default property settings.
12. To get rid of all existing layout properties, delete the WebGrid from the web form designer.
13. From the Toolbox, drag and drop an UltraWebGrid onto the designer form. Set the WebGrid position and size as desired.
14. Click on the WebGrid and set the following properties:

DataSource = DataSet11
DataMember = Customers

15. Add the following code to the page load event to load the default layout XML if it is found:

In Visual Basic:

If Me.IsPostBack = False Then

```
' load default layout for both appearance and behavior
Dim strFilePath As String
Dim xmlReader As System.Xml.XmlTextReader
Try
  Try
    strFilePath = Me.MapPath("/UltraWebGridv2vs2002" _
      & "/Save_Load_Layouts_XML" _
      & "/DefaultLayout.xml")
```

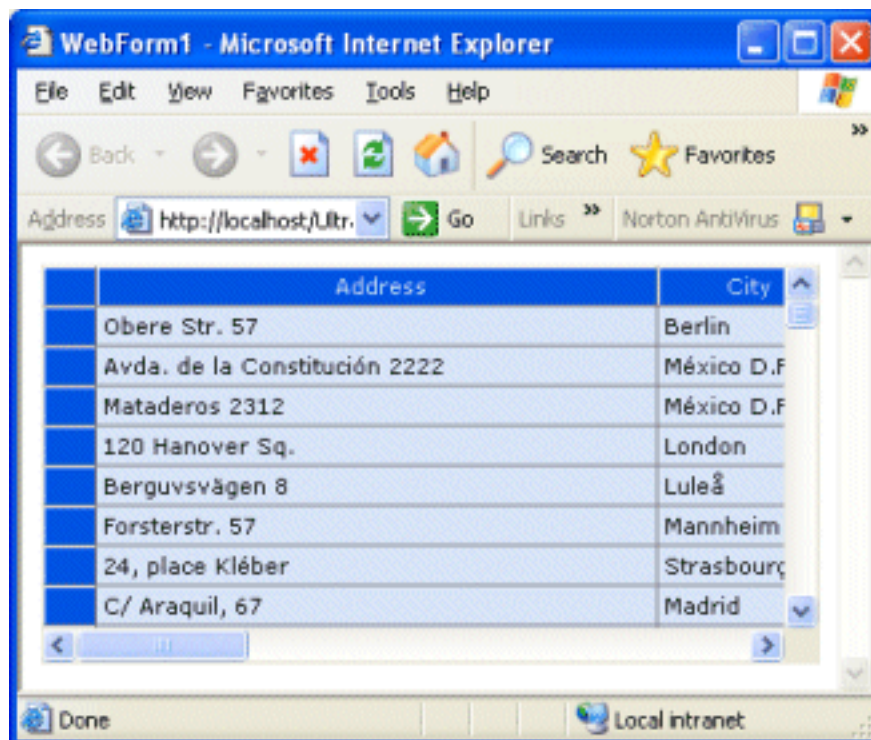
```

        xmlReader = New System.Xml.XmlTextReader( _
            strFilePath)
    Catch
        strFilePath = Me.MapPath("/Save_Load_Layouts_XML" _
            & "/DefaultLayout.xml")
        xmlReader = New System.Xml.XmlTextReader( _
            strFilePath)
    End Try
    UltraWebGrid1.DisplayLayout.LoadLayout(xmlReader, True, True, False, False)
    xmlReader.Close()
Catch ex As Exception
    Console.WriteLine(ex.Message.ToString)
End Try

' fill dataset and bind to grid
Me.OleDbDataAdapter1.Fill(DataSet1)
Me.UltraWebGrid1.DataBind()
End If

```

16. Run the project and observe the appearance settings from the `DisplayLayout.xml` file were loaded and applied to the WebGrid:



Stop the project.

You can also save layouts with code by invoking the **SaveLayout** method of the `DisplayLayout` object of the WebGrid.

Review

This tutorial introduces the concepts associated with saving and loading WebGrid layouts to and from XML files.

Create WebGrid With Code

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

It is possible to create an instance of the WebGrid completely from code. This capability is used on web forms where a variable number of WebGrid controls are presented based on program logic. After creating the grid the grid attributes can be set to position the grid on the resulting web page.

Questions

- How can I create a WebGrid completely with code?
- How do I set the absolute position of a code generated grid?

Solution

An instance of a WebGrid can be created in code with the following statements. Set the grid Attributes.CssStyle properties to position the grid on the web form.

Sample Project

The sample project creates a WebGrid in code, adds some columns, and adds some row data.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. In the Solution Explorer right-click the References node and select Add Reference... From the .NET tab select the following references:

```
Infragistics.WebIU.Shared  
Infragistics.WebUI.UltraWebGrid
```

Click OK.

3. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

4. Add the following code to the page's **Load** event. This code creates a new web grid control, adds two columns, adds five rows with two cells per row, applies some borders, finds a reference to the form on the page, and adds the WebGrid to pages the Form1 controls collection.

In Visual Basic:

```
' declare and create a new WebGrid  
Dim uwgSample As New _  
Infragistics.WebUI.UltraWebGrid.UltraWebGrid("WebGrid1")  
  
' add two columns  
uwgSample.Columns.Add("Col1", "Column One")
```

```

uwgSample.Columns.Add("Col2", "Column Two")

' add 5 rows
Dim intPtr As Integer
For intPtr = 1 To 5
    Dim uwgRow As New Infragistics.WebUI.UltraWebGrid.UltraGridRow()

    ' add cells to the row
    Dim uwgCell As New Infragistics.WebUI.UltraWebGrid.UltraGridCell()
    uwgCell.Text = "Col1 value " + CStr(intPtr)
    uwgRow.Cells.Add(uwgCell)

    uwgCell = New Infragistics.WebUI.UltraWebGrid.UltraGridCell()
    uwgCell.Text = "Col2 value " + CStr(intPtr)
    uwgRow.Cells.Add(uwgCell)

    ' add row to grid
    uwgSample.Rows.Add(uwgRow)
Next

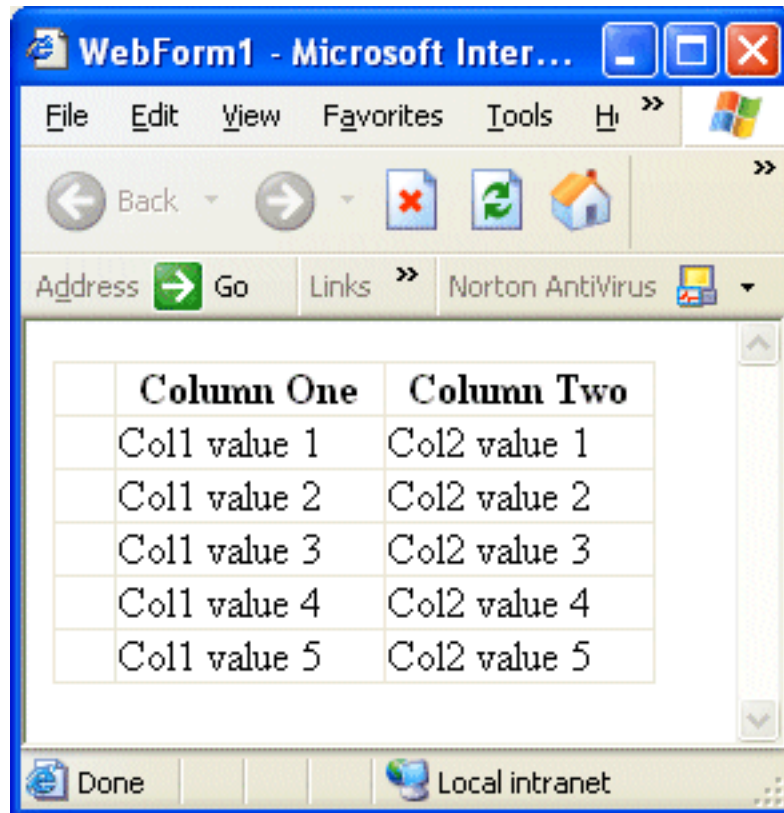
' make it look pretty
uwgSample.DisplayLayout.RowStyleDefault.BorderStyle _
= BorderStyle.Solid
uwgSample.DisplayLayout.RowStyleDefault.BorderWidth _
= New System.Web.UI.WebControls.Unit(1)

' find Form1
Dim ctlForm1 As System.Web.UI.HtmlControls.HtmlForm
Dim ctlTest As System.Web.UI.Control
For Each ctlTest In Page.Controls
    Try
        If ctlTest.ID = "Form1" Then
            ctlForm1 = ctlTest
            Exit For
        End If
    Catch
    End Try
Next

' add the WebGrid to the form
If Not ctlForm1 Is Nothing Then
    ctlForm1.Controls.Add(uwgSample)
End If

```

5. Run the project and observe the grid on the web page:



Stop the project.

6. Extend this project by setting the absolute position of the grid on the web page. Add the following lines of code to the page's **Load** event handler:

In Visual Basic:

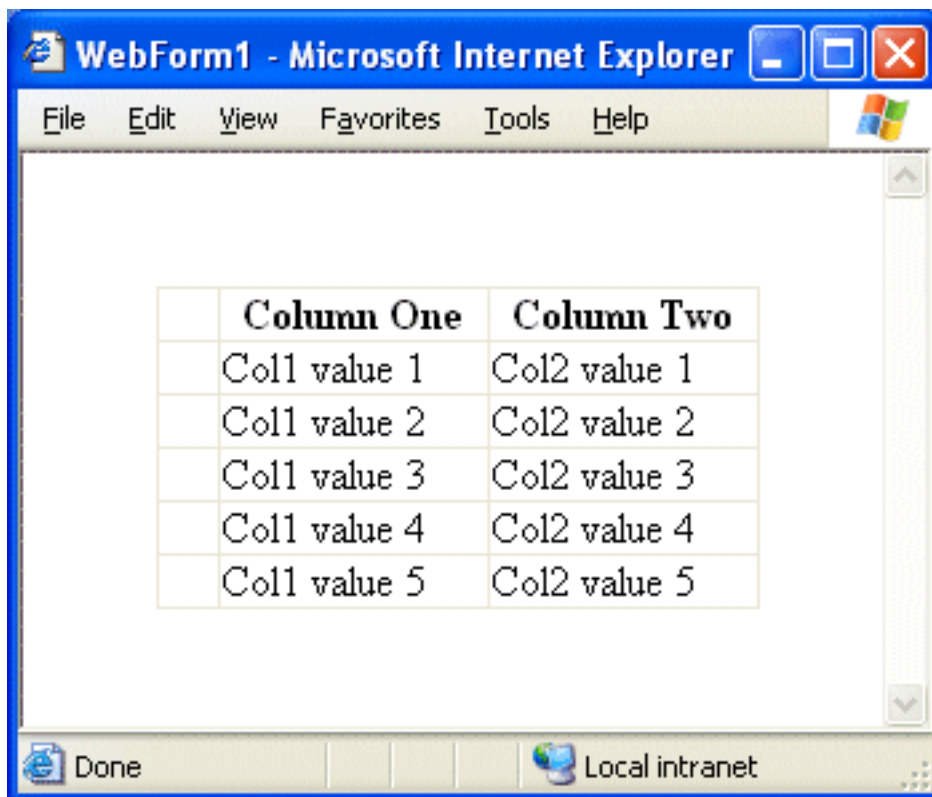
```
' make it look pretty
uwgSample.DisplayLayout.RowStyleDefault.BorderStyle _
= BorderStyle.Solid
uwgSample.DisplayLayout.RowStyleDefault.BorderWidth _
= New System.Web.UI.WebControls.Unit(1)

' set grid position on form
uwgSample.Attributes.CssStyle("POSITION") = "absolute"
uwgSample.Attributes.CssStyle("LEFT") = "50px"
uwgSample.Attributes.CssStyle("TOP") = "50px"

' find Form1
Dim ctlForm1 As System.Web.UI.HtmlControls.HtmlForm
Dim ctlTest As System.Web.UI.Control
For Each ctlTest In Page.Controls
. . .
```

Note It is VERY important that you use the correct case on the `CssStyle` property keys. THEY ARE CASE SENSITIVE!

7. Run the project and you should see the grid offset by 50 pixels from the top and left:



Review

This tutorial shows how to create a WebGrid using code. The process consists of creating a new instance of the grid, adding some columns, adding some cells to some rows and adding the rows to the grid, retrieving a reference to the form on the page and adding the WebGrid to the form controls collection. Also illustrated is the setting of the CssStyle Attributes of the grid to place it at a specific position on the web form.

Update Database With Row Changes

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Updating the database with row changes can be a challenge in the web forms environment, and there is not any one methodology that meets the needs of all applications. Issues of state will vary depending on the type of data entry being performed and the size of the underlying data.

This tutorial is a little more robust than most and includes error trapping and user message code.

Questions

- How do I allow the user to enter changes, add new rows, delete rows, and apply the changes to the underlying database?

Solution

In the **InitializeLayout** event, tell the grid to allow AddNew, Delete and Update, and show the AddNew box. Implement event handlers for the **UpdateCell**, **AddRow**, and **DeleteRow** events.

Sample Project

This sample project displays the Northwind Customers table and allows the user to update, delete and add rows.

To create this project, perform the following steps:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
6. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

7. Click on the WebGrid and set the following properties:

DataSource = DataSet11

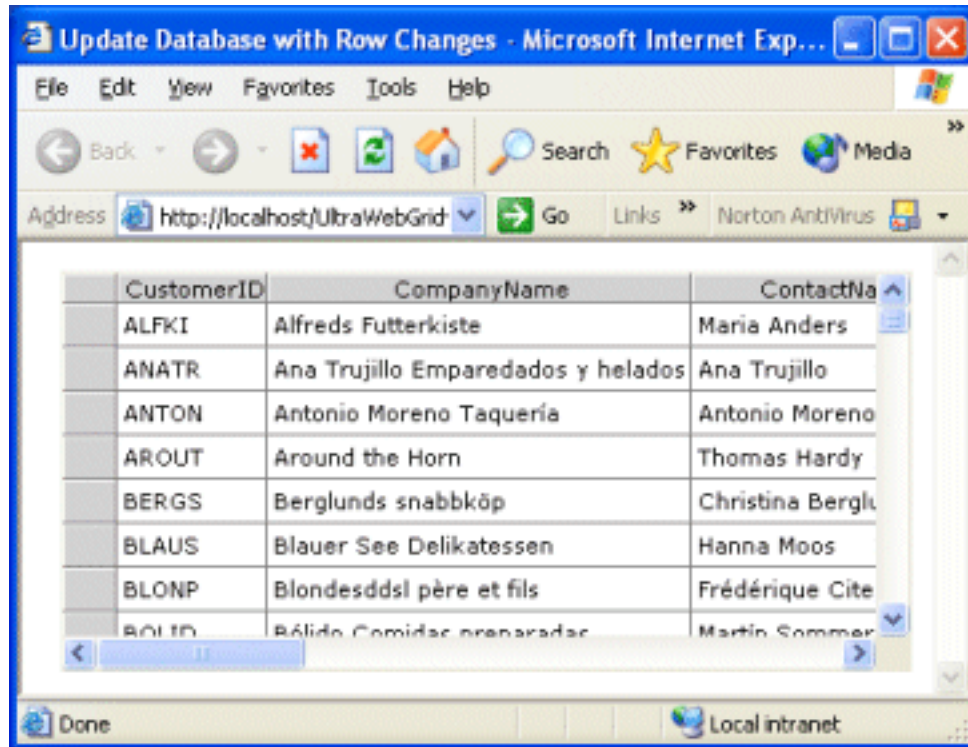
DataMember = Customers

8. Add the following code to the page's **Load** event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

9. Run the project and you should see something like:

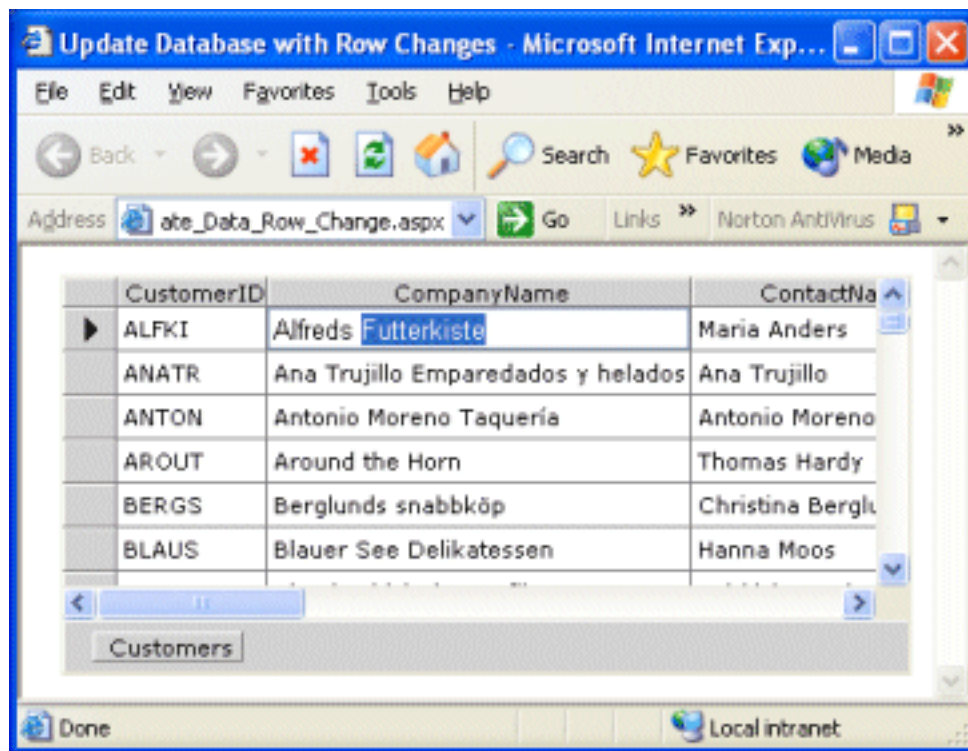


10. This tutorial allows for cell updates, row deletes, row adds, and the AddNewBox needs to be visible. Add the following code to the WebGrid **InitializeLayout** event:

In Visual Basic:

```
e.Layout.AllowAddNewDefault _
= Infragistics.WebUI.UltraWebGrid.AllowAddNew.Yes
e.Layout.AllowDeleteDefault _
= Infragistics.WebUI.UltraWebGrid.AllowDelete.Yes
e.Layout.AllowUpdateDefault _
= Infragistics.WebUI.UltraWebGrid.AllowUpdate.Yes
e.Layout.AddNewBox.Hidden = False
e.Layout.Bands(0).DataKeyField="CustomerID"
```

11. Run the project and you should now see something like the following with the AddNewBox displayed at the bottom of the WebGrid:



Notice that you can now update cells by double-clicking on them, delete rows by selecting rows and pressing the delete key, and add new rows by selecting anything in the WebGrid and clicking the AddNewBox.

- This tutorial responds to the **UpdateCell** event, thus the WebGrid will perform a post back every time a cell is changed. For some applications this many post backs may be undesirable and the developer should look at using the **UpdateCellBatch** event instead. (Batch mode updating is covered by the [Update Database In Batch Mode](#) tutorial.)

To apply cell updates, add the following code to the WebGrid **UpdateCell** event handler:

In Visual Basic:

```
' declare a new customers data table
Dim dtCustomers As New DataTable()

' try to retrieve a dataset for the row containing the changed cell
Try
    OleDbDataAdapter1.SelectCommand.CommandText &= _
        " WHERE " _
        & UltraWebGrid1.Bands(0).DataKeyField _
        & " = '" _
        & e.Cell.Row.DataKey.ToString _
        & "'"
    OleDbDataAdapter1.Fill(dtCustomers)
Catch ex As Exception
    DisplayMessages(ex.Message.ToString _
        , OleDbDataAdapter1.SelectCommand.CommandText.ToString)
End Try

' check to make sure you have a record
If dtCustomers.Rows.Count <= 0 Then
    DisplayMessages("Original record not found, key= " _
        & e.Cell.Row.DataKey.ToString _
        , OleDbDataAdapter1.SelectCommand.CommandText.ToString)
End If
```

```

' attempt to update the cell value
Try
    Dim drCustomer As DataRow
    drCustomer = dtCustomers.Rows(0)
    drCustomer(e.Cell.Column.Key) = e.Cell.Value
Catch ex As Exception
    DisplayMessages(ex.Message.ToString _
        , e.Cell.Column.Key, e.Data.ToString)
End Try

' attempt to send update to database
Try
    OleDbDataAdapter1.Update(dtCustomers)
Catch ex As Exception
    DisplayMessages(ex.Message.ToString)
End Try

' set data key if key field changed
If e.Cell.Column.Key = UltraWebGrid1.Bands(0).DataKeyField Then
    e.Cell.Row.DataKey = e.Cell.Value
End If

```

13. A convenient method of displaying detected error messages in the browser is needed. One option is to add a method which will take up to four messages and place them in session variables where they will be accessible by another web form and redirect the response to that form.

Add the following **DisplayMessages** method to WebForm1.aspx:

In Visual Basic:

```

Private Sub DisplayMessages(ByVal v_strMessage1 As String _
, Optional ByVal v_strMessage2 As String = "" _
, Optional ByVal v_strMessage3 As String = "" _
, Optional ByVal v_strMessage4 As String = "")
    Session.Add("Message1", v_strMessage1)
    Session.Add("Message2", v_strMessage2)
    Session.Add("Message3", v_strMessage3)
    Session.Add("Message4", v_strMessage4)
    Response.Redirect("Messages.aspx")
End Sub

```

14. Add a new WebForm named Messages.aspx. On this form add four text boxes and set the following properties of each:
- Height** = 90px
TextMode = MultiLine
15. Arrange the text boxes on the form as needed.
16. Run the project, double-click on one of the cells, change the content, and click on a different row. Notice the grid performs a post back to the server. Terminate and restart the project and notice the change you made is now permanent.
17. When you select a row on the grid and press Delete a post back is made to the server to fire the **DeleteRow** event. The code in this event retrieves the original record, deletes the row and updates the database.
18. Add the following code to the WebGrid **DeleteRow** event to delete the selected record from the database:

In Visual Basic:

```
' declare a new customers data table
Dim dtCustomers As New DataTable()

' try to retrieve a dataset for the row being deleted
Try
    OleDbDataAdapter1.SelectCommand.CommandText &= _
        " WHERE " _
        & UltraWebGrid1.Bands(0).DataKeyField _
        & " = '" _
        & e.Row.Cells.FromKey( _
        UltraWebGrid1.Bands(0).DataKeyField).Value.ToString _
        & "'"
    OleDbDataAdapter1.Fill(dtCustomers)
Catch ex As Exception
    DisplayMessages(ex.Message.ToString _
        , OleDbDataAdapter1.SelectCommand.CommandText.ToString)
End Try

' check to make sure you have a record
If dtCustomers.Rows.Count <= 0 Then
    DisplayMessages("Original record not found!" _
        , OleDbDataAdapter1.SelectCommand.CommandText.ToString)
End If

' attempt to delete the row
Try
    dtCustomers.Rows(0).Delete()
Catch ex As Exception
    DisplayMessages("Delete Row Failed!", ex.Message.ToString)
End Try
```

Review

This exercise demonstrates how to enable the updating of data in the WebGrid. The user may add, delete or modify data on the client, and any changes made will be posted back to the server and reflected in the underlying datasource.

Update Database In Batch Mode

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Updating the database with row changes can be a challenge in the web forms environment, and there is no single methodology that meets the needs of all applications. Issues of state will vary depending on the type of data entry being performed and the size of the underlying data.

This tutorial uses the batch events of the WebGrid, is a little more robust than most and includes error trapping and user messaging code.

Questions

- My application can not afford to process a post back on each cell change, row delete and row add. How can I update all user changes in a single batch update?

Solution

Implement the Batch versions of the update cell, delete row and add row events. These events can be a little more challenging than the non-batch events when it comes to adding multiple new rows in a single postback. But in terms of responsiveness for the end user, the reduced postback advantage far outweighs the additional complexity.

Sample Project

This sample project displays the Northwind Customers table and allows the user to update, delete and add rows.

To create this project, perform the following steps:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.

6. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

7. Click on the WebGrid and set the following properties:

DataSource = DataSet11

DataMember = Customers

8. Add the following code to the page's **Load** event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

9. To use the Batch events for cell updates, row deletes and row adds, a post back needs to be invoked. For the purposes of this tutorial, add a button to the WebForm1.aspx designer and set the following properties:

Text = Submit **ID** = btnSubmit

10. To configure the WebGrid for updates, deletes and adds, set the default cell click action to edit, and add a hidden GUID column to assist with multiple new row adds, add the following code to the WebGrid **InitializeLayout** event:

In Visual Basic:

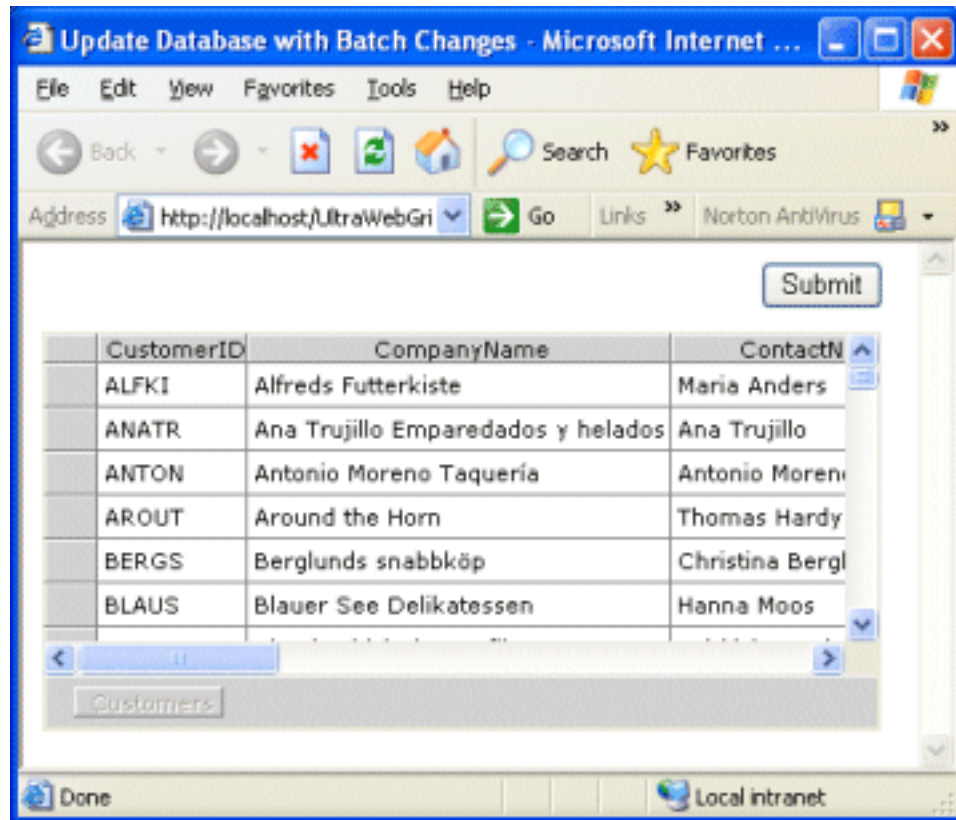
```
' set the DataKeyField so when the events are thrown the
' row.datakey property can be populated
e.Layout.Bands(0).DataKeyField = "CustomerID"

' turn on edit, update, delete and show the AddNewBox
e.Layout.AllowAddNewDefault _
= Infragistics.WebUI.UltraWebGrid.AllowAddNew.Yes
e.Layout.AllowDeleteDefault _
= Infragistics.WebUI.UltraWebGrid.AllowDelete.Yes
e.Layout.AllowUpdateDefault _
= Infragistics.WebUI.UltraWebGrid.AllowUpdate.Yes
e.Layout.AddNewBox.Hidden = False

' set the default cell click action to edit
e.Layout.CellClickActionDefault = _
Infragistics.WebUI.UltraWebGrid.CellClickAction.Edit

' add a hidden GUID column to band(0)
With e.Layout.Bands(0)
    .Columns.Add("GUID")
    .Columns.FromKey("GUID").Hidden = True
End With
```

11. Run the project and you should see something like:



12. The project needs a place to collect new rows being added to the underlying data. Add the following class scope collection declaration:

In Visual Basic:

```
Dim c_colNewCustomerDataRows As New Collection()
```

13. Each of the events (**UpdateCellBatch**, **DeleteRowBatch**, and **AddRowBatch**) needs to test to see if the data set has been retrieved from the database. Add the following private method to retrieve the data set and add a GUID column for use when identifying new rows:

In Visual Basic:

```
Private Sub LoadUpdatableDataSet()
    ' retrieve dataset from server if not already available
    If DataSet11.Customers.Rows.Count <= 0 Then
        ' fill dataset with server data
        OleDbDataAdapter1.Fill(DataSet11)

        ' add guid column for new rows
        Dim colGUID As New DataColumn("GUID", GetType(String))
        DataSet11.Customers.Columns.Add(colGUID)
    End If
End Sub
```

14. To apply updates coming from the grid, add code to the **UpdateCellBatch** event. This event is fired for each cell updated in the grid and performs the following steps:

- Checks to see if the customers data table is already loaded and if it is not the data adapter is used to fill the customers data table with rows from the data source.
- If there is no DataKey value for the row being updated, you know it is a new row being added and the GUID value of the new row is used to retrieve the new row from the new

rows collection.

- Or, uses the **DataKey** property of the event argument to find the row being updated.
- If the data row can not be found the message form is invoked.
- The updated data value is placed in the appropriate column of the data row.

Add the following code to the WebGrid **UpdateCellBatch** event:

In Visual Basic:

```
Dim drCustomers As DataRow

' retrieve dataset from server if not already available
LoadUpdatableDataSet()

' test to see if this cell goes into a new row
If e.Cell.Row.DataKey Is Nothing Then
    ' look in the collection for the new row
    drCustomers = CType(c_colNewCustomerDataRows.Item( _
        e.Cell.Row.Cells.FromKey("GUID").Value.ToString), DataRow)
Else
    ' try to find row containing the changed cell
    Try
        drCustomers = DataSet11.Customers.FindByCustomerID( _
            e.Cell.Row.DataKey.ToString)
    Catch ex As Exception
        DisplayMessages("Original record not found for updating, key= " _
            & e.Cell.Row.DataKey.ToString)
    End Try
End If

' attempt to update the cell value
Try
    drCustomers(e.Cell.Column.Key) = e.Cell.Value
Catch ex As Exception
    DisplayMessages(ex.Message.ToString _
        , e.Cell.Column.Key, e.Data.ToString)
End Try
```

15. To add the new rows and apply the updates to the underlying database after all updates, deletes and adds have been applied and rebind the dataset to the WebGrid if new rows were added, add the following code to the page's **Unload** event:

In Visual Basic:

```
' test to see if there new rows to add
If c_colNewCustomerDataRows.Count > 0 Then
    Dim drCustomers As DataSet1.CustomersRow
    For Each drCustomers In c_colNewCustomerDataRows
        DataSet11.Customers.Rows.Add(drCustomers)
    Next
End If

' attempt to send updates to database
Try
    OleDbDataAdapter1.Update(DataSet11.Customers)
Catch ex As Exception
    DisplayMessages(ex.Message.ToString)
```

```
End Try
```

```
' if new rows were added, then rebind  
If c_colNewCustomerDataRows.Count > 0 Then  
    UltraWebGrid1.DataBind()  
End If
```

16. To delete the rows in the database that were deleted by the user on the WebGrid, add code to the **DeleteRowBatch** event. This event is fired for each row deleted by the user and performs the following steps:

- Retrieve the Customers data table using the data adapter if not already available.
- Use the DataKey property supplied with the event to find the row deleted by the user.
- Display a message if the row can not be found (probably deleted by another user).
- Invoke the Delete method of the row to have the row deleted in the database with the data table updates are persisted.

Add the following code to the **DeleteRowBatch** event handler:

In Visual Basic:

```
' retrieve dataset from server if not already available  
LoadUpdatableDataSet()  
  
' try to find row to delete  
Dim drCustomers As DataRow  
Try  
    drCustomers = DataSet11.Customers.FindByCustomerID( _  
        e.Row.DataKey.ToString)  
Catch ex As Exception  
    DisplayMessages("Original record not found for deleting, key = " _  
        & e.Row.DataKey.ToString)  
End Try  
  
If drCustomers Is Nothing Then  
    DisplayMessages("Original record not found for deleting, key = " _  
        & e.Row.DataKey.ToString)  
End If  
  
' attempt to delete the row  
Try  
    drCustomers.Delete()  
Catch ex As Exception  
    DisplayMessages("Delete failed!" _  
        , ex.Message.ToString _  
        , e.Row.DataKey.ToString)  
End Try
```

17. A convenient method of displaying messages in the browser is needed. One option is to add a method which will take up to four messages and place them in a text box on the form. The code below adds this method, called **DisplayMessages**.

Add the following DisplayMessages method to WebForm1.aspx:

In Visual Basic:

```
Private Sub DisplayMessages(ByVal v_strMessage1 As String _
```

```

, Optional ByVal v_strMessage2 As String = "" _
, Optional ByVal v_strMessage3 As String = "" _
, Optional ByVal v_strMessage4 As String = "")

txtMessages.Text += v_strMessage1 + vbCrLf + vbCrLf

If v_strMessage2 <> "" Then txtMessages.Text _
+= v_strMessage2 + vbCrLf + vbCrLf

If v_strMessage3 <> "" Then txtMessages.Text _
+= v_strMessage3 + vbCrLf + vbCrLf

If v_strMessage4 <> "" Then txtMessages.Text _
+= v_strMessage4 + vbCrLf + vbCrLf

End Sub

```

18. Add a text box to the form and set the properties as follows:

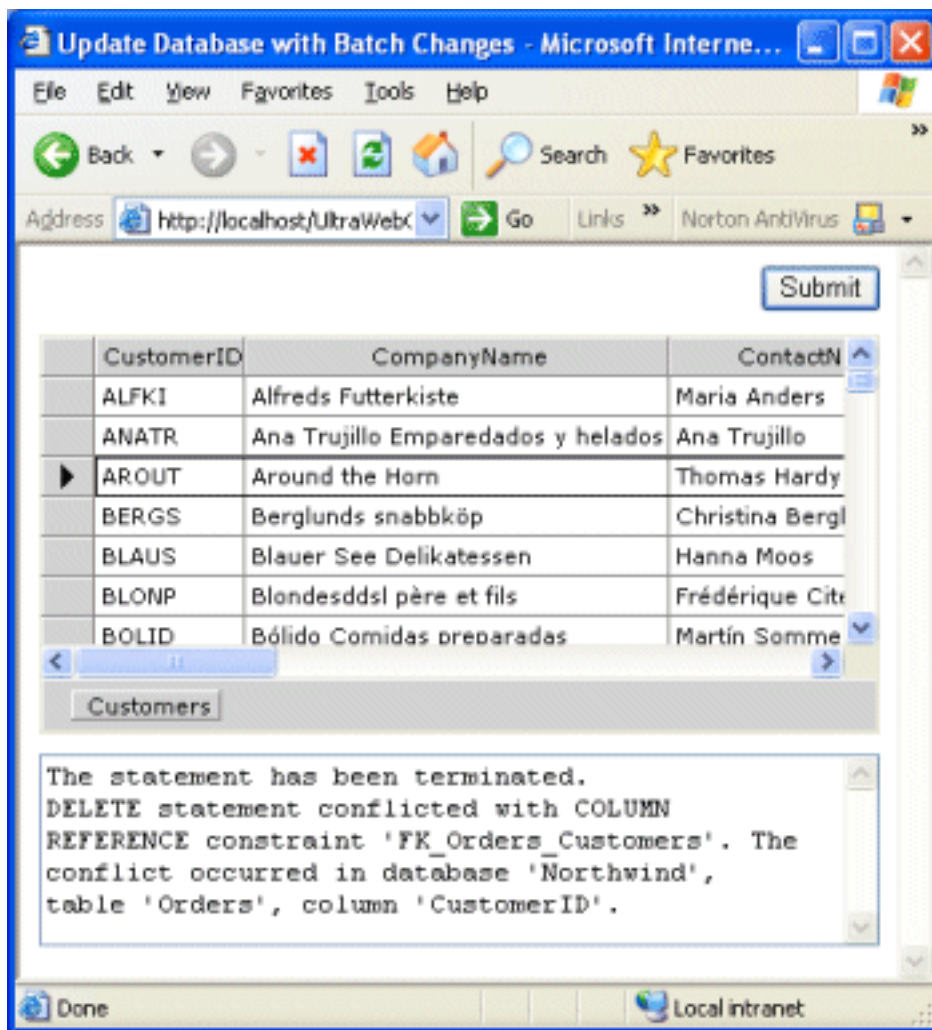
```

ID = txtMessages
Height = 160px
TextMode = MultiLine

```

19. Arrange the text box on the form as needed.

20. Run the project, select one of the rows in the WebGrid, press Delete on the keyboard and the row will be deleted from the grid. Click Submit and the project will attempt to delete the row. You should expect an error to occur because deleting any of the customers in the Northwind database will probably violate a database constraint (Customer records are all linked to Order records.) You should expect to see something like:



21. To add the new rows to the data table that were added by the user, you must add code to the WebGrid **AddRowBatch** event. This event is fired for each new row added by the user, so if the user added three new rows, this event will fire three times before the **UpdateCellBatch** event fires at all. The real challenge here is to be able to create new rows, get them populated with user entered cell values, add them to the data table, and have any constraints such as unique primary keys satisfied. Since you are working with the Northwind Customers data table which contains a user entered primary key, you are presented with some unique challenges:
- If the user adds multiple new rows before submitting the page, how do you suspend these new rows until the cell values are populated?
 - When the **UpdateCellBatch** event fires, how do you know which row to put the values into since the primary key value is user entered and may not be populated yet?

To satisfy this requirement, this tutorial declares a class scope data row collection to contain the new rows, creates a new row each time the **AddRowBatch** event fires, sets a GUID value in the new row and this same GUID value in the WebGrid row that caused the event to fire. Now you have a GUID in each new row and the same GUID in each new row of the grid, so when the **UpdateCellBatch** event fires, you can use the GUID from the hidden column to find the appropriate row in the new rows collection.

In the WebGrid **AddRowBatch** event you must perform these steps:

- Retrieve the dataset from the server if it is not already available.
- Creates a new blank data row and sets the GUID value. Adds this new blank row to the new rows collection for later population of cell values in the UpdateCellBatch event.
- Sets the GUID in the WebGrid row hidden GUID column.

The following code implements the desired behavior:

In Visual Basic:

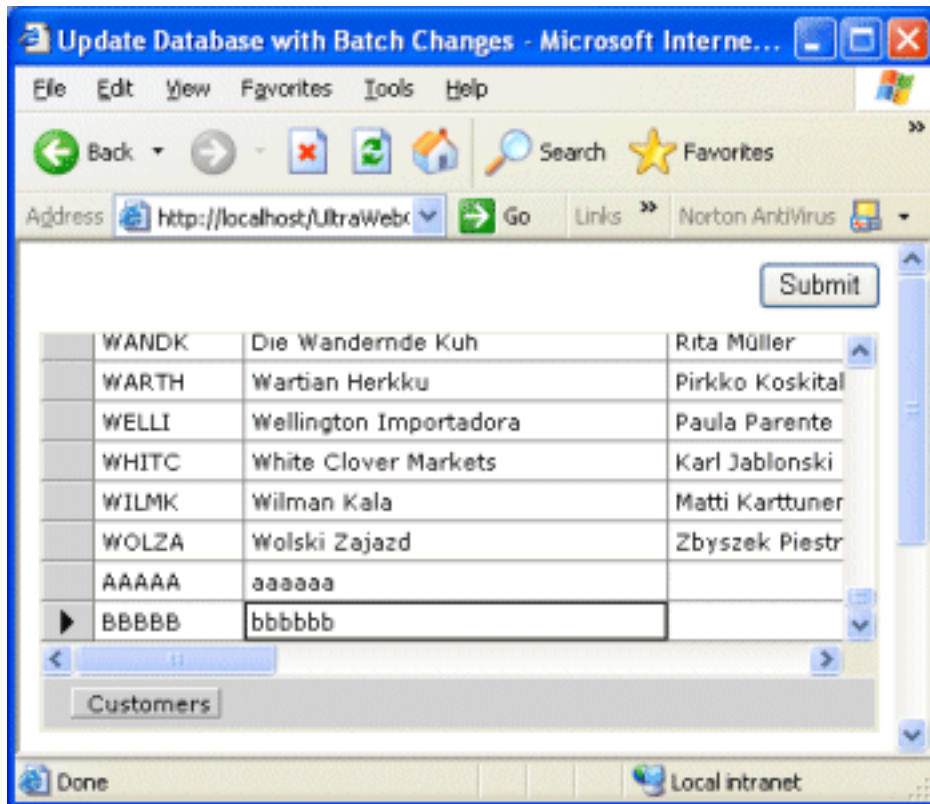
```
' retrieve dataset from server if not already available
LoadUpdatableDataSet()

' create a new blank row and set the GUID
Dim drCustomers As DataRow = DataSet11.Customers.NewRow
drCustomers("GUID") = Guid.NewGuid.ToString("N")

' add new blank row to new rows collection
c_colNewCustomerDataRows.Add(drCustomers _
, drCustomers("GUID").ToString)

' set the GUID in the WebGrid row
e.Row.Cells.FromKey("GUID").Value = drCustomers("GUID")
```

22. Run the project. Click on one of the cells in the grid to set focus and click the Customers AddNewButton. Select the CustomerID field and type a unique CustomerID value (remember this is the unique primary key column) and type something into the company name column. Repeat the add procedure a couple of times. Click Submit and you should see something like:



Review

This project shows how to use the batch update, delete and add events of the WebGrid. Particular emphasis is placed on a methodology for adding multiple new rows to a data table where the user can enter the unique primary key value.

Data Paging (Basic)

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Because of bandwidth and other considerations, Paging is a common methodology used to present the user with a small portion of a total data set. The WebGrid has a built-in paging feature which can be implemented quickly and provides the capabilities needed for many smaller applications.

Questions

- I have a relatively small set of data that needs to be paged by the letters of the alphabet. What is the quickest way to implement this paging using the WebGrid?

Solution

Use the WebGrid paging features to implement basic paging.

Sample Project

This project uses the Northwind Customers table as a source of name and address information and the WebGrid built-in paging logic. To create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Customers and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
6. Click the Infragistics.WebUI.Shared reference and set the following properties:
CopyLocal = True
7. Click on the WebGrid and set the following properties:

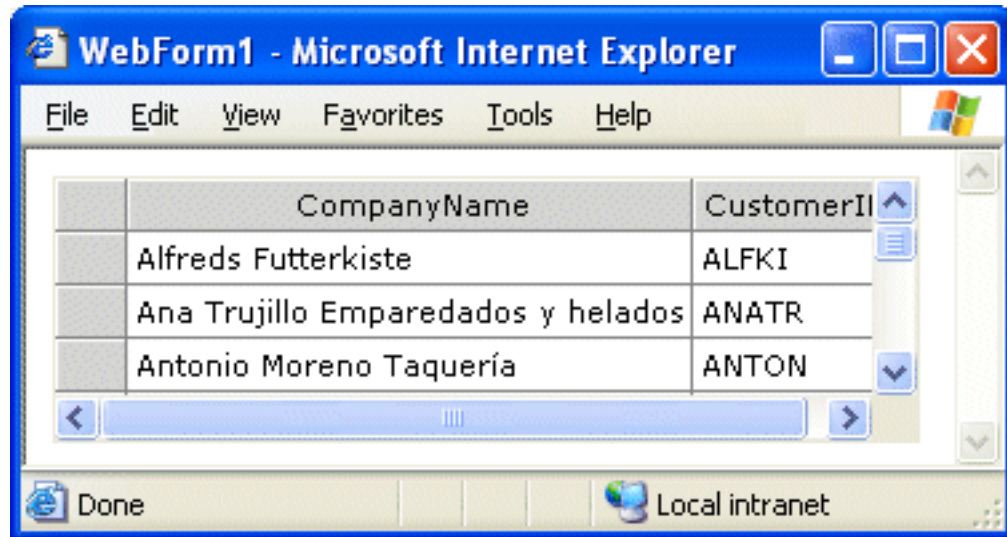
DataSource = DataSet11
DataMember = Customers

8. Add the following code to the page's **Load** event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

9. Run the project and depending on which version of the Northwind database you connect to you should see something like:



10. Click the WebGrid and set the following properties:

DisplayLayout.Pager.AllowCustomPaging = True
DisplayLayout.Pager.AllowPaging = True
DisplayLayout.Pager.StyleMode = CustomLabels

DisplayLayout.EnableInternalRowsManagement = False
DisplayLayout.ScrollBar = Never

Height = (remove this value)

11. Add the following class scope declaration for the alphabetic characters:

In Visual Basic:

```
Private strAlphabet() As String = {"A", "B", "C" _  
    , "D", "E", "F", "G", "H", "I", "J", "K", "L", "M" _  
    , "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W" _  
    , "X", "Y", "Z" }
```

12. Add the following private method to retrieve and bind data to the grid:

In Visual Basic:

```
Private Sub RetrieveAndBindData(ByVal v_strLetter As String)  
    OleDbDataAdapter1.SelectCommand.CommandText _  
    = "SELECT CustomerID, CompanyName" _
```

```

    & " FROM Customers WHERE (CustomerID LIKE '" _
    & v_strLetter _
    & "%' )"

    ' retrieve and bind data
    OleDbDataAdapter1.Fill(DataSet11)
    UltraWebGrid1.DataBind()
End Sub

```

13. In the page's **Load** event handler, replace the code you added earlier with:

In Visual Basic:

```

' set customer pager labels
UltraWebGrid1.DisplayLayout.Pager.CustomLabels _
= strAlphabet

' only do this on first page load
If Me.IsPostBack = False Then
    ' retrieve and bind first page
    RetrieveAndBindData("A")
End If

```

14. Add the following code to the WebGrid PageIndexChanged event handler:

In Visual Basic:

```

' retrieve and bind data for selected page
RetrieveAndBindData(strAlphabet(e.NewPageIndex - 1))

```

15. Add the following code to the page's **PreRender** event to set the custom labels and page count:

In Visual Basic:

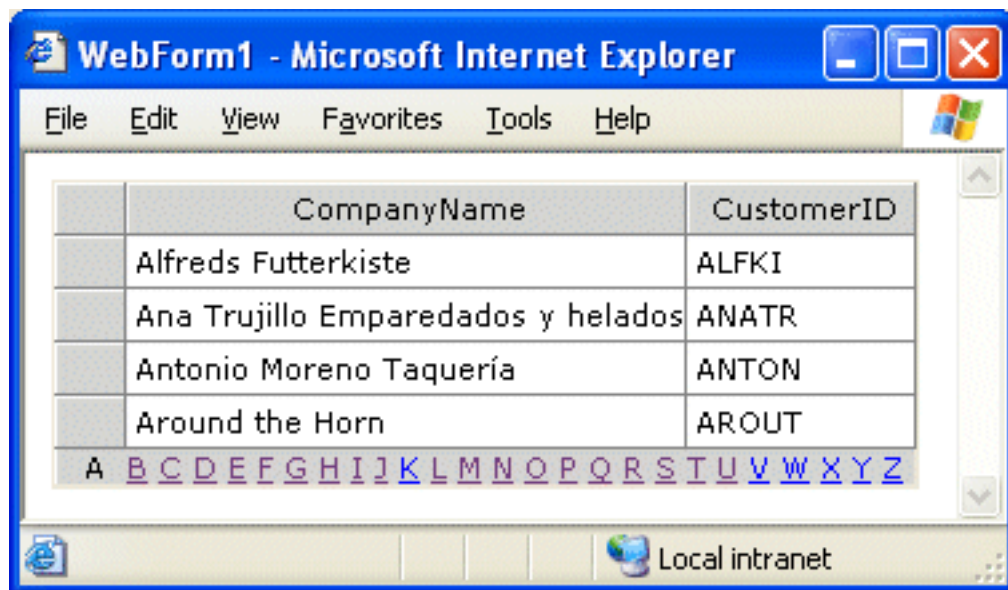
```

' set custom pager labels
UltraWebGrid1.DisplayLayout.Pager.CustomLabels _
= strAlphabet

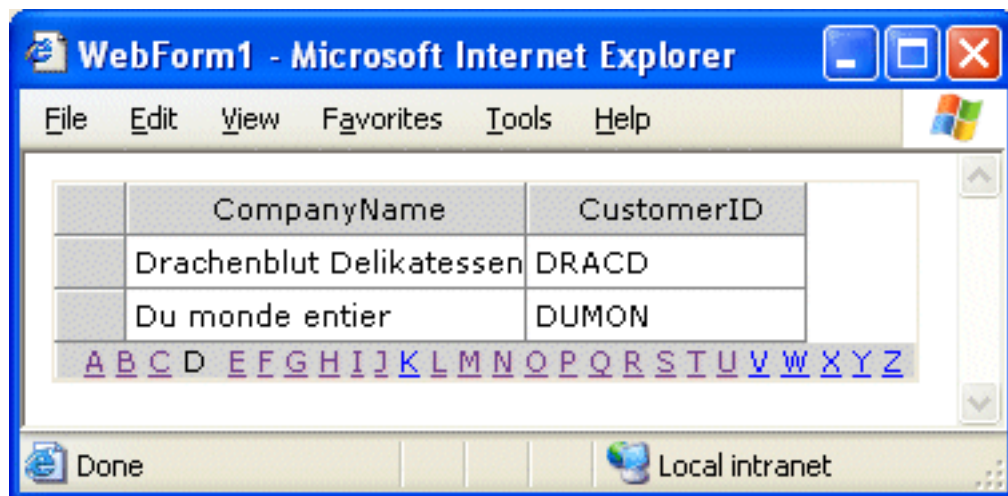
' set the page count
UltraWebGrid1.DisplayLayout.Pager.PageCount = 26

```

16. Run the project and you should see something like:



17. Click on one of the page buttons and you should see something like:



Stop the project.

Review

This tutorial shows how to use the built-in paging features to create a simple paging application.

Data Paging (Advanced)

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

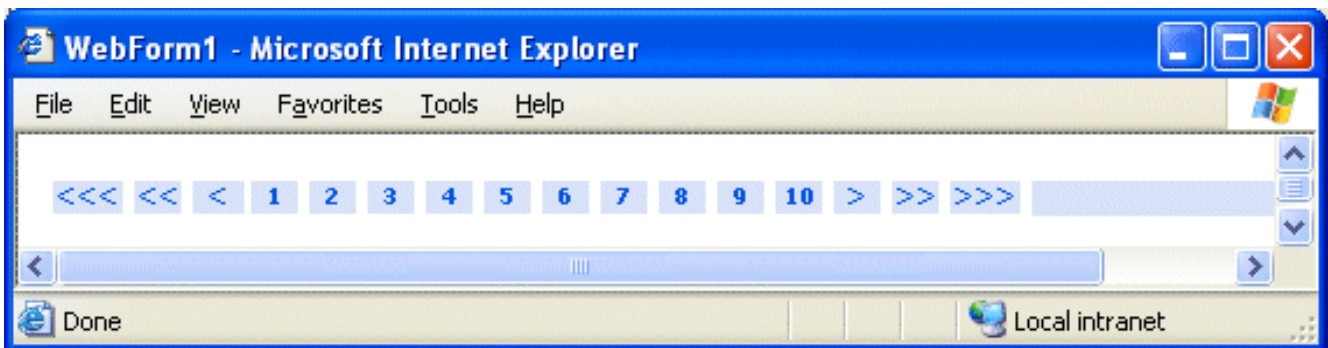
Background

No matter how much bandwidth is available, Data Paging is a must. There are search and data selection instances when the user will select thousands of records. It is not practical in the Web Forms environment to package up thousands of records and send them to the client. (From a practical standpoint, presenting thousands of records simultaneously to any user is likely to be overkill.)

Data Paging is currently the most effective methodology for dealing with selections of large quantities of records. There are many different ways to present paging to the user; however it is generally accepted that the interface should include the following:

- First Page Link
- Group Back Link
- Previous Page Link
- A Group of page Links
- Next Page Link
- Group Forward Link
- Last Page Link

And display something like:



Some developers put images in the control button to enhance the user experience.

Questions

- My users have search and select capabilities allowing them to select thousands of records. How do I implement this in a way that allows for small pages of data to be presented to the user and minimize server state?

Solution

One way to approach this is to retrieve a data table of keys to the records selected, and add this low overhead data table to the session state. Then implement a paging methodology allowing the user to quickly access the page of data they are looking for.

Sample Project

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`

- Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
 4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
 5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
 6. Click the Infragistics.WebUI.Shared reference and set the following properties:
 - CopyLocal** = True
 7. Click on the WebGrid and set the following properties:
 - DataSource** = DataSet11
 - DataMember** = OrderDetails
 - DisplayLayout.ScrollBar** = Never
 - Height** = *(remove this setting)*
 8. Add the following code to the page's **Load** event to fill the dataset and bind it to the WebGrid:

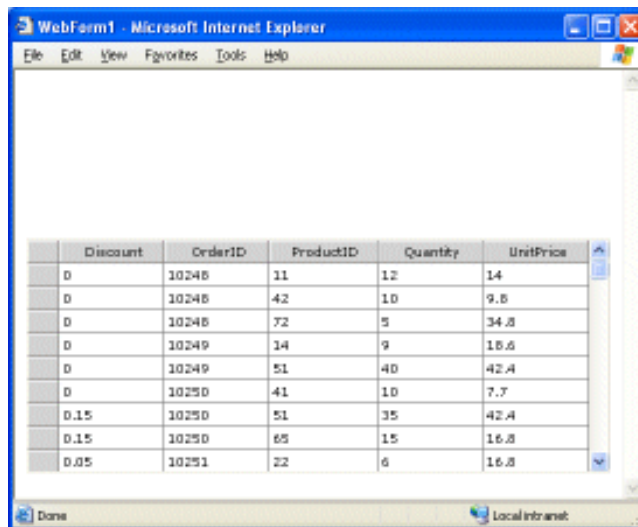
In Visual Basic:

```

If Me.IsPostBack = False Then
    Me.OleDbSelectCommand1.CommandText _
    = "SELECT TOP 100 * FROM [Order Details]"
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If

```

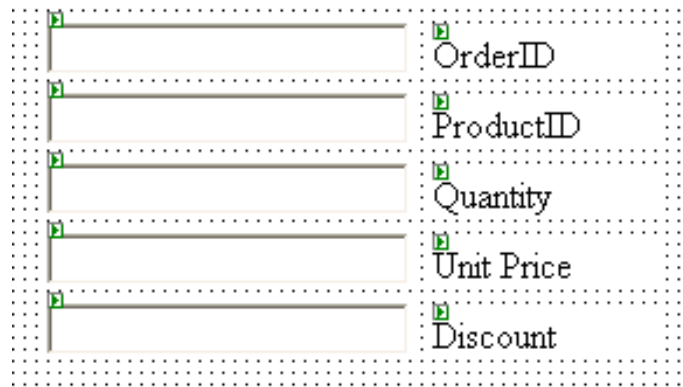
9. Run the project and depending on which version of the Northwind database you connect to you should see something like:



Stop the project.

10. Right-click the grid, click the Columns (Collection) property and open the Columns Collection Editor. Use the Up/Down arrows to arrange the columns as follows:
 - OrderID
 - ProductID
 - Quantity
 - UnitPrice
 - Discount
11. For this tutorial, add 5 web forms text boxes and labels above the WebGrid. Set the label text as in the following

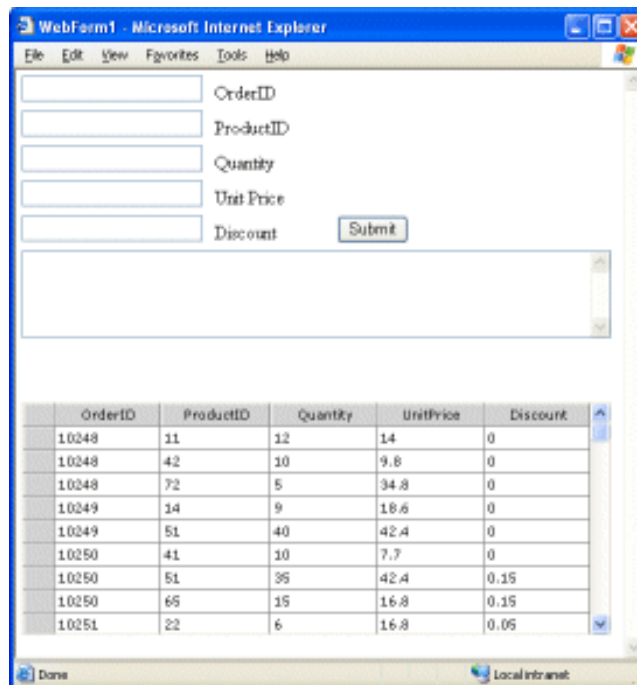
illustration. Change the **ID** property of the text boxes to txt followed by the name of their corresponding column and arrange them something like:



12. Add a web forms button to the right of the Discount label. Change the **ID** property to btnSubmit and the Text property to Submit.
13. Add a text box below the selection text boxes to display the SQL select command and/or error messages. Click on this text box and set the following properties:

ID = txtMessage
TextMode = MultiLine

14. Run the project and you should see something like:



Stop the project.

15. On the initial page load, this tutorial performs a search with no search criteria and displays page 1. Replace the code added to the page's **Load** event with:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.PerformSearch()
    Me.BindToGrid(1)
End If
```

16. The page load references the **PerformSearch** and **BindToGrid** methods which you must now write. **PerformSearch** - PerformSearch retrieves the user input, creates a select command to retrieve the appropriate records, performs the query and places the resulting data table into a session variable. Add the following PerformSearch method to the WebForm1.aspx code behind:

In Visual Basic:

```
Private Sub PerformSearch()  
    Dim sbSQL As New System.Text.StringBuilder(4096)  
    Dim blnAddAnd As Boolean = False  
    Dim blnRunSelection As Boolean = False  
    Dim dtAccount As DataTable  
  
    ' build select part of SQL  
    sbSQL.Append("SELECT OrderID, ProductID FROM [Order Details] WHERE")  
  
    ' order id selection  
    If Trim(txtOrderID.Text) <> "" Then  
        If blnAddAnd = True Then sbSQL.Append(" AND")  
        sbSQL.Append(" OrderID LIKE '%" _  
            & Trim(txtOrderID.Text) & "%'" & vbCrLf)  
        blnRunSelection = True  
        blnAddAnd = True  
    End If  
  
    ' product id selection  
    If Trim(txtProductID.Text) <> "" Then  
        If blnAddAnd = True Then sbSQL.Append(" AND")  
        sbSQL.Append(" ProductID LIKE '%" _  
            & Trim(txtProductID.Text) & "%'" & vbCrLf)  
        blnRunSelection = True  
        blnAddAnd = True  
    End If  
  
    ' Quantity selection  
    If Trim(txtQuantity.Text) <> "" Then  
        If blnAddAnd = True Then sbSQL.Append(" AND")  
        sbSQL.Append(" Quantity LIKE '%" _  
            & Trim(txtQuantity.Text) & "%'" & vbCrLf)  
        blnRunSelection = True  
        blnAddAnd = True  
    End If  
  
    ' UnitPrice selection  
    If Trim(txtUnitPrice.Text) <> "" Then  
        If blnAddAnd = True Then sbSQL.Append(" AND")  
        sbSQL.Append(" UnitPrice LIKE '%" _  
            Trim(txtUnitPrice.Text) + "%'" + vbCrLf)  
        blnRunSelection = True  
        blnAddAnd = True  
    End If  
  
    ' Discount selection  
    If Trim(txtDiscount.Text) <> "" Then  
        If blnAddAnd = True Then sbSQL.Append(" AND")  
        sbSQL.Append(" Discount LIKE '%" _  
            & Trim(txtDiscount.Text) & "%'" & vbCrLf)  
        blnRunSelection = True  
        blnAddAnd = True  
    End If  
  
    ' if no selection, retrieve all  
    If blnRunSelection = False Then  
        sbSQL.Append(" OrderID IS NOT NULL AND ProductID IS NOT NULL" & vbCrLf)  
    End If  
  
    ' put SQL into display  
    txtMessage.Text = sbSQL.ToString & vbCrLf  
  
    ' put SQL into select command text and fill data set  
    Try  
        Me.OleDbSelectCommand1.CommandText = sbSQL.ToString  
        Me.OleDbDataAdapter1.Fill(DataSet11)  
    Catch ex As Exception  
        txtMessage.Text += ex.Message  
    End Try
```

```

' put selection data table into session state
Session.Add("SelectionTable", DataSet11.Tables("Order Details"))
End Sub

```

BindToGrid - BindToGrid takes a page number parameter, invokes the PopulatePage method and binds the returned data table to the grid DataSource and invokes the DataBind method of the WebGrid. This tutorial will add more code to this method later to assist with paging. Add the following method to the WebForm1.aspx code behind:

In Visual Basic:

```

Private Sub BindToGrid(ByVal v_intPage As Integer)
' retrieve the selection table
Dim dtSelection As DataTable _
= CType(Session("SelectionTable"), DataTable)

' populate page and bind to grid
UltraWebGrid1.DataSource _
= PopulatePage(dtSelection, v_intPage, c_intPageSize)
UltraWebGrid1.DataBind()
End Sub

```

17. PopulatePage uses the selection data table, requested page number, and page size to create SQL select command text to retrieve the records for the requested page from the data base. The SQL text is applied to the select command and a new data table filled with the results. The CurrentPage, TotalPages and TotalRows session variables are set and the data set returned. Add the following method to the WebForm1.aspx code behind:

In Visual Basic:

```

Private Function PopulatePage(ByVal v_dt As DataTable _
, ByVal v_intPage As Integer _
, ByVal v_intPageSize As Integer) As DataSet

Dim sbSQL As New System.Text.StringBuilder()
Dim intStartPtr As Integer
Dim intEndPtr As Integer
Dim intTotalPages As Integer
Dim intPtr As Integer

sbSQL.Append(" SELECT * FROM [Order Details] WHERE" & vbCrLf)

' set start and end pointers
intStartPtr = (v_intPage - 1) * v_intPageSize
intEndPtr = intStartPtr + v_intPageSize - 1
If intEndPtr > v_dt.Rows.Count - 1 Then intEndPtr = v_dt.Rows.Count - 1
intTotalPages = CInt(Fix((v_dt.Rows.Count - 1) / v_intPageSize)) + 1

' pass rows and add to where clause
For intPtr = intStartPtr To intEndPtr
If intPtr <> intStartPtr Then sbSQL.Append(" OR")
sbSQL.Append(" (OrderID=" & v_dt.Rows(intPtr).Item("OrderID").ToString)
sbSQL.Append(" AND ProductID=" _
& v_dt.Rows(intPtr).Item("ProductID").ToString & ")" & vbCrLf)
Next

' put SQL into text box
txtMessage.Text += sbSQL.ToString & vbCrLf

' set select command text and fill dataset
Me.OleDbSelectCommand1.CommandText = sbSQL.ToString
Dim dtSelected As New DataTable("Order Details")
Me.OleDbDataAdapter1.Fill(dtSelected)
DataSet11.Tables.Remove("Order Details")
DataSet11.Tables.Add(dtSelected)

' set current page session variable
Session.Add("CurrentPage", v_intPage)
Session.Add("TotalPages", intTotalPages)
Session.Add("TotalRows", CInt(v_dt.Rows.Count))

```



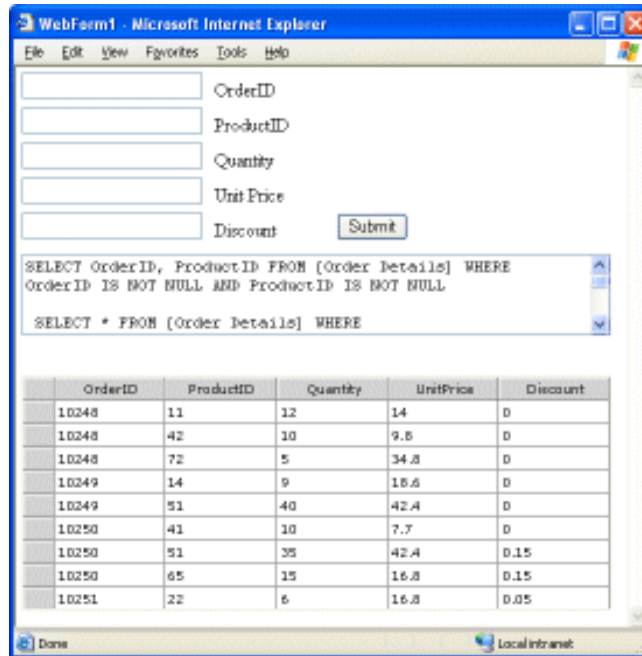
```
Return (DataSet1)
End Function
```

18. Add the following code to the btnSubmit button control's **Click** event to invoke the PerformSearch and BindToGrid methods when the user clicks the Submit button:

In Visual Basic:

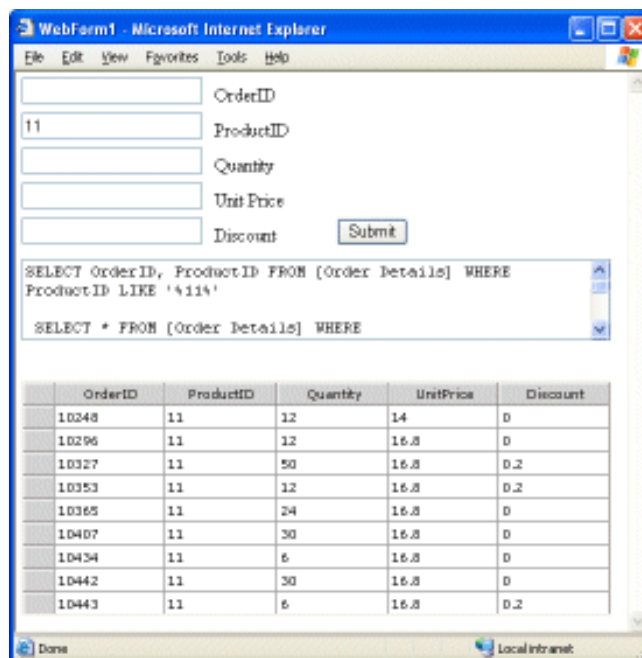
```
Me.PerformSearch()
Me.BindToGrid(1)
```

19. Run the Project and you should see something like:



Notice there are two SQL select statements displayed: The first is the select statement used to select all of the possible entries to display, and the second is the select statement used to retrieve the rows belonging to page 1.

You haven't added the paging code yet, but you can select different data by entering a value in one of the text boxes and clicking Submit. The following example shows the display after entering 11 in the Product text box and pressing Submit:

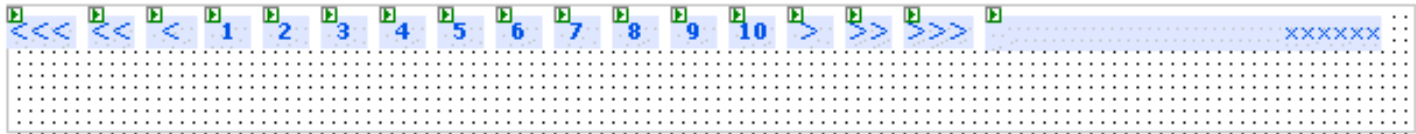


Stop the project.

20. Rather than use the built-in paging, which restricts the form and function of our paging buttons, this advanced tutorial will use a Web User Control to contain a row of buttons and provide button manipulation methods.
Select from the main menu Project, Add Web User Control Change the file name to PageLinks.ascx and click Open.
21. From the Toolbox, HTML tab; drag a Grid Layout Panel to the PageLinds.ascx designer and size it to about 55 pixels height by 600 pixels width.
22. From the Toolbox, Web Forms tab; drag and drop a Button control on the grid layout panel. Click on this button and set the following properties:

ID = btnFirst
BackColor = InactiveCaptionText
BorderStyle = None
Font.Bold = True
Font.Name = Verdana
Font.Size = XX-Small
ForeColor = ActiveCaption
Text = "<<<"

23. The following cut-out from Visual Studio shows what you are trying to accomplish:



24. Copy the first button and paste it to create each new button, changing the properties as follows:

1. Name = btnPreviousGroup Text = "<<<"	9. Name = btn7 Text = "7"
2. Name = btnPrevious1 Text = "<<"	10. Name = btn8 Text = "8"
3. Name = btn1 Text = "1"	11. Name = btn9 Text = "9"
4. Name = btn2 Text = "2"	12. Name = btn10 Text = "10"
5. Name = btn3 Text = "3"	13. Name = btnNext1 Text = ">"
6. Name = btn4 Text = "4"	14. Name = btnNextGroup Text = ">>"
7. Name = btn5 Text = "5"	15. Name = btnLast Text = ">>>"
8. Name = btn6 Text = "6"	

25. From the Toolbox, Web Forms tab; drag a drop a Label on the grid layout panel and set the following properties:

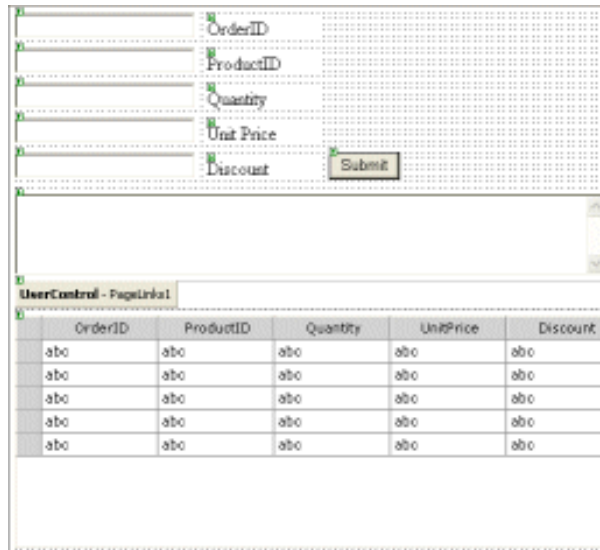
ID = lblTotalPages
BackColor = InactiveCaptionText
Font.Name = Verdana
Font.Size = XX-Small
ForeColor = ActiveCaption

26. Set position and size of all controls to look something like the above. Now you are ready to add the user control to the form designer. Right-click WebForm1.aspx and select View Designer.
27. According to Microsoft, by design the UserControl in Visual Studio 2002 is not able to deal with absolute positioning. So if you just drag the control to the form it is going to show up at the top of the form. This tutorial places the UserControl

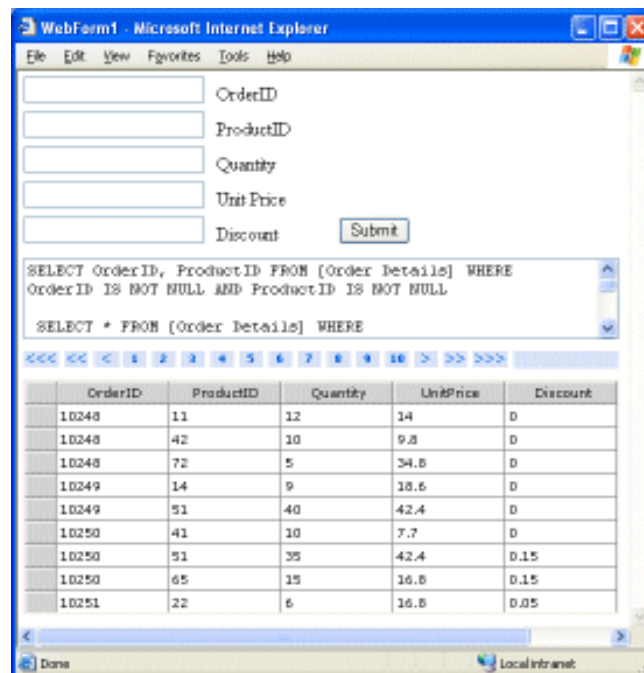
in a Panel on the web form designer to make it position properly.

From the Toolbox, Web Forms tab; Drag and drop a Panel onto the web form designer. Position it between the messages text box and the WebGrid and make it wide enough to allow the page links to display. Click the panel, wait a second, click it again and remove the Panel text.

28. With the panel container in place, drag PageLinks.ascx from the Solution Explorer and drop it on the panel. The designer should look something like:



29. Run the project and you should see something like:



Stop the project.

30. A property setting is needed in the user control to allow the container form to set the text property of lblTotalPages. Select the PageLinks.ascx code editor and add the following public property:

In Visual Basic:

```
Public Property TotalPagesText() As String
    Get
        Return (lblTotalPages.Text)
    End Get

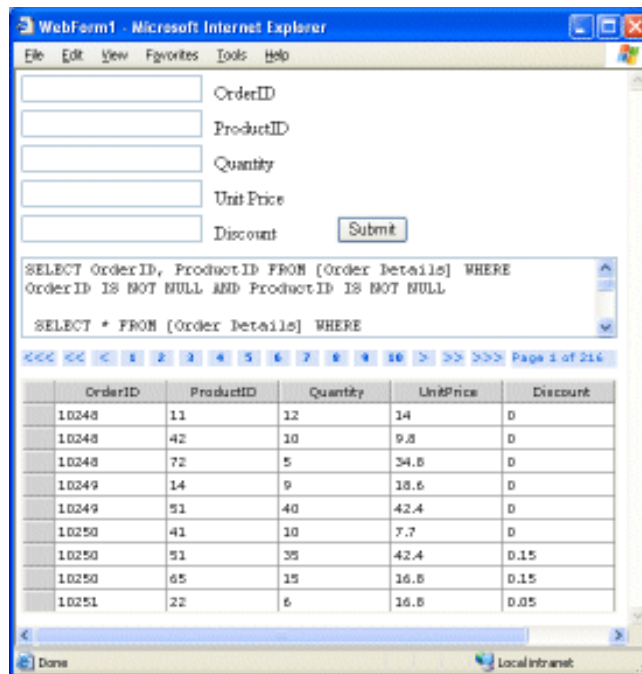
    Set(ByVal Value As String)
        lblTotalPages.Text = Value
    End Set
End Property
```

31. Next, you will add some code to the web form code to set the TotalPagesText property. Visual Studio 2002 does not add any reference to the web form to the user control when it is added, so you need to go to the top of the web form code and add the following along with all the other Protected WithEvents declarations:
`Protected WithEvents PageLinks1 As PageLinks`
32. Add the following code to the bottom of the BindToGrid method:

In Visual Basic:

```
' Set properties of paging links control
PageLinks1.TotalPagesText = " Page " _
& Session("CurrentPage").ToString _
& " of " _
& Session("TotalPages").ToString
```

33. Run the project and you should see something like:



Notice the Page 1 of 216 text in the total pages label. Then stop the project.

34. Next you must add code to the UserControl to set the text, visibility and enabled properties of the buttons base on the page being displayed. Add the following private method to the PageLinks.ascx.vb file to set the text and visibility of the page buttons:

In Visual Basic:

```
Private Sub SetPageButton( _
ByVal v_btn As System.Web.UI.WebControls.Button _
, ByVal v_intNumber As Integer _
, ByVal v_intFirstPageNumber As Integer _
, ByVal v_intCurrentPage As Integer _
, ByVal v_intTotalPages As Integer)

    With v_btn
        ' set text of button
        .Text = (v_intFirstPageNumber + v_intNumber - 1).ToString

        ' set enabled state of button
        If v_intCurrentPage _
= (v_intFirstPageNumber + v_intNumber - 1) Then
            .Enabled = False
        Else
            .Enabled = True
        End If
    End With
End Sub
```

```

        ' set visible state of button
    If (v_intFirstPageNumber + v_intNumber - 1) _
    > v_intTotalPages Then
        .Visible = False
    Else
        .Visible = True
    End If
End With
End Sub

```

35. Add the following public method to the PageLinks.ascx.vb file to set the text, visibility and enabled properties of all of the page links buttons:

In Visual Basic:

```

Public Sub SetButtons(ByVal v_intCurrPage As Integer _
, ByVal v_intTtlPgs As Integer)

    Dim intFirstPage As Integer
    intFirstPage = CInt(Fix((v_intCurrPage - 1) / 10) * 10 + 1)

    With btnPreviousGroup
        If v_intTtlPgs <= 10 Or v_intCurrPage <= 10 Then
            .Enabled = False
        Else
            .Enabled = True
        End If
    End With

    With btnPrevious1
        If v_intCurrPage <= 1 Then
            .Enabled = False
        Else
            .Enabled = True
        End If
    End With

    SetPageButton(btn1, 1, intFirstPage, v_intCurrPage, v_intTtlPgs)
    SetPageButton(btn2, 2, intFirstPage, v_intCurrPage, v_intTtlPgs)
    SetPageButton(btn3, 3, intFirstPage, v_intCurrPage, v_intTtlPgs)
    SetPageButton(btn4, 4, intFirstPage, v_intCurrPage, v_intTtlPgs)
    SetPageButton(btn5, 5, intFirstPage, v_intCurrPage, v_intTtlPgs)
    SetPageButton(btn6, 6, intFirstPage, v_intCurrPage, v_intTtlPgs)
    SetPageButton(btn7, 7, intFirstPage, v_intCurrPage, v_intTtlPgs)
    SetPageButton(btn8, 8, intFirstPage, v_intCurrPage, v_intTtlPgs)
    SetPageButton(btn9, 9, intFirstPage, v_intCurrPage, v_intTtlPgs)
    SetPageButton(btn10, 10, intFirstPage, v_intCurrPage, v_intTtlPgs)

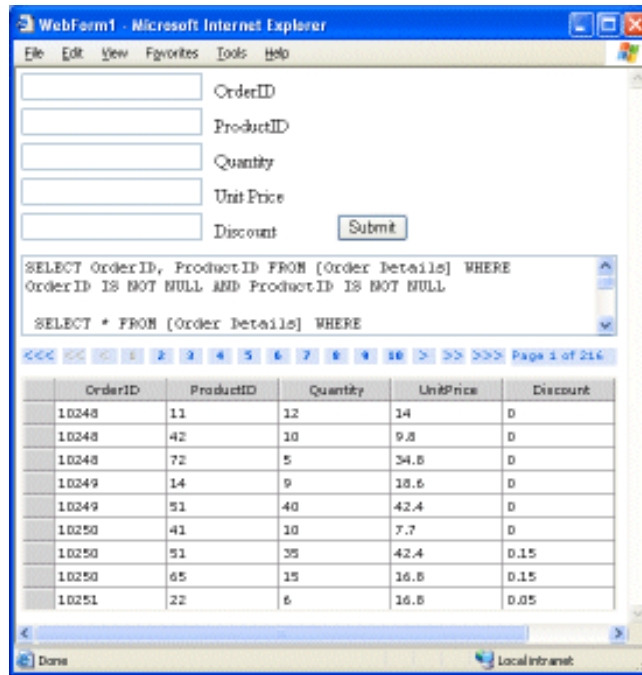
    With btnNext1
        If v_intCurrPage >= v_intTtlPgs Then
            .Enabled = False
        Else
            .Enabled = True
        End If
    End With

    With btnNextGroup
        If v_intCurrPage + 10 < v_intTtlPgs Then
            .Enabled = True
        Else
            .Enabled = False
        End If
    End With

    ' set session variable to remember first page in set
    Session.Add("FirstPage", intFirstPage)
End Sub

```

36. Run the project and you should see something like:



Notice the enabled property of some of the buttons has changed. Stop the project.

37. Now the tutorial is complete to the point that the page link buttons are displayed. You are now ready to react to users clicking the buttons. The event handlers are located in the PageLinks UserControl and subsequently raise an event to the container which is the web form. Define the event to be raised by placing the following line of code above any methods in the PageLinks.ascx.vb file:

```
Event LinkButtonClick(ByVal PageSelect As Integer)
```

38. Add event handlers for all of the link buttons by adding the following region of code to the PageLinks.ascx.vb file:

In Visual Basic:

```
#Region " Navigation Button Event Handlers"
Private Sub btnFirst_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnFirst.Click
RaiseEvent LinkButtonClick(1)
End Sub

Private Sub btnPreviousGroup_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnPreviousGroup.Click
RaiseEvent LinkButtonClick(CInt(Session("CurrentPage")) - 10)
End Sub

Private Sub btnPrevious1_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnPrevious1.Click
RaiseEvent LinkButtonClick(CInt(Session("CurrentPage")) - 1)
End Sub

Private Sub btn1_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn1.Click
RaiseEvent LinkButtonClick(CInt(Session("FirstPage")))
End Sub

Private Sub btn2_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn2.Click
RaiseEvent LinkButtonClick(CInt(Session("FirstPage")) + 1)
End Sub

Private Sub btn3_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
```

```

Handles btn3.Click
    RaiseEvent LinkButtonClick(CInt(Session("FirstPage")) + 2)
End Sub

Private Sub btn4_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn4.Click
    RaiseEvent LinkButtonClick(CInt(Session("FirstPage")) + 3)
End Sub

Private Sub btn5_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn5.Click
    RaiseEvent LinkButtonClick(CInt(Session("FirstPage")) + 4)
End Sub

Private Sub btn6_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn6.Click
    RaiseEvent LinkButtonClick(CInt(Session("FirstPage")) + 5)
End Sub

Private Sub btn7_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn7.Click
    RaiseEvent LinkButtonClick(CInt(Session("FirstPage")) + 6)
End Sub

Private Sub btn8_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn8.Click
    RaiseEvent LinkButtonClick(CInt(Session("FirstPage")) + 7)
End Sub

Private Sub btn9_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn9.Click
    RaiseEvent LinkButtonClick(CInt(Session("FirstPage")) + 8)
End Sub

Private Sub btn10_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn10.Click
    RaiseEvent LinkButtonClick(CInt(Session("FirstPage")) + 9)
End Sub

Private Sub btnNext1_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnNext1.Click
    RaiseEvent LinkButtonClick(CInt(Session("CurrentPage")) + 1)
End Sub

Private Sub btnNextGroup_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnNextGroup.Click
    RaiseEvent LinkButtonClick(CInt(Session("CurrentPage")) + 10)
End Sub

Private Sub btnLast_Click( _
ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnLast.Click
    RaiseEvent LinkButtonClick(CInt(Session("TotalPages")))
End Sub
#End Region

```

39. Now that the UserControl will be raising an event when the user clicks on a link button, the host form needs to respond to this event. Add the following event handler to the web form code to handle the LinkButtonClick event from the PageLinks user control:

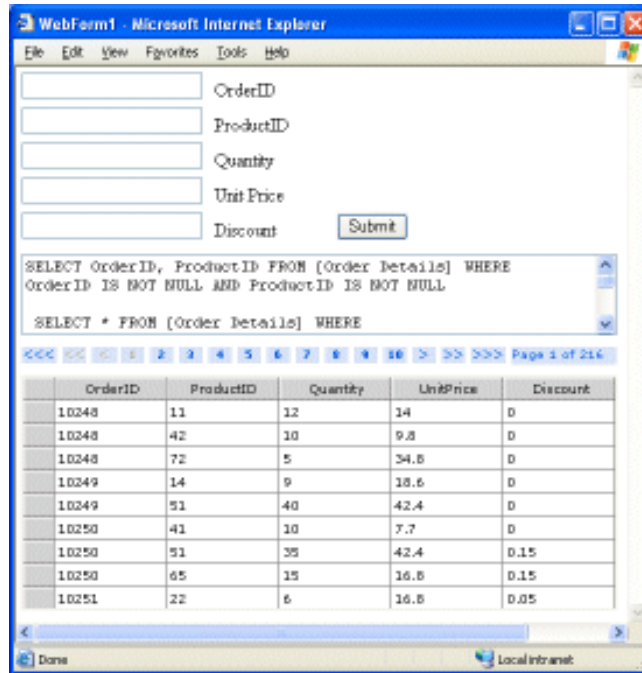
In Visual Basic:

```

Private Sub PageLinks1_LinkButtonClick( _
ByVal PageSelect As Integer) _
Handles PageLinks1.LinkButtonClick
    Me.BindToGrid(PageSelect)
End Sub

```

40. Run the project and you should see something like:



Click on the page navigation buttons and observe the results. Change the selection and click Submit and observe the results. Then stop the project.

41. This project is currently set to display only 10 records in the WebGrid. Change the number of records to display in the WebGrid to 25 by changing the following declaration in the web form code:

In Visual Basic:

```
Dim c_intPageSize As Integer = 25
```

Review

This tutorial shows how to create an advanced paging application. The developer performs step-by-step enhancements to the project and is exposed to the following methodologies:

- Dynamic creation of an SQL SELECT statement based on user input.
- A methodology for dealing with thousands of selected records in a Web Forms environment.
- A Web User Control containing page navigation buttons. Communications between the Web Form and Web User Control.

Load On Demand

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

When displaying hierarchical data in the grid it is easy to create performance problems with the child band rows that are not visible. To address this issue, the WebGrid supports Load on Demand which means that when the user clicks on an expansion indicator the grid performs a post back to the server to retrieve the rows of the child band.

Questions

- I am having bandwidth issues with a hierarchical WebGrid which has many rows in the child bands that are not displayed until the user requests them. How can I populate the band data when the user clicks on the expansion indicator?

Solution

Use the Load on Demand feature of the WebGrid to improve performance by uploading only the visible records as needed.

Sample Project

This sample project hierarchically binds the WebGrid to the Northwind Customers and Orders data tables and uses Load on Demand features to populate the Orders records when the user clicks on the expansion indicator.

1. Start a new Web Forms project.
2. To add the Customers table: From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Customers and click Close, click (All Columns) and click OK.
 - Click Finish.
3. To add the Orders table: From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select the same connection as in the previous step.
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Orders and click Close, click (All Columns) and click OK.

- Click Finish.
- From the main Menu select Data, Generate Dataset and click OK From Solution Explorer, open DataSet1.xsd and you should see both the Customers and Orders tables listed. Drag the bottom of each table down until you can see all of the columns. Right-click the CustomerID column in the Customers table and select Add, New Relation Change the Child element to Orders and click OK. You have now setup the relationship between the Customers and Orders tables that make the relationship hierarchical.
 - From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
 - In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
 - Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True
 - Click on the WebGrid and set the following properties:

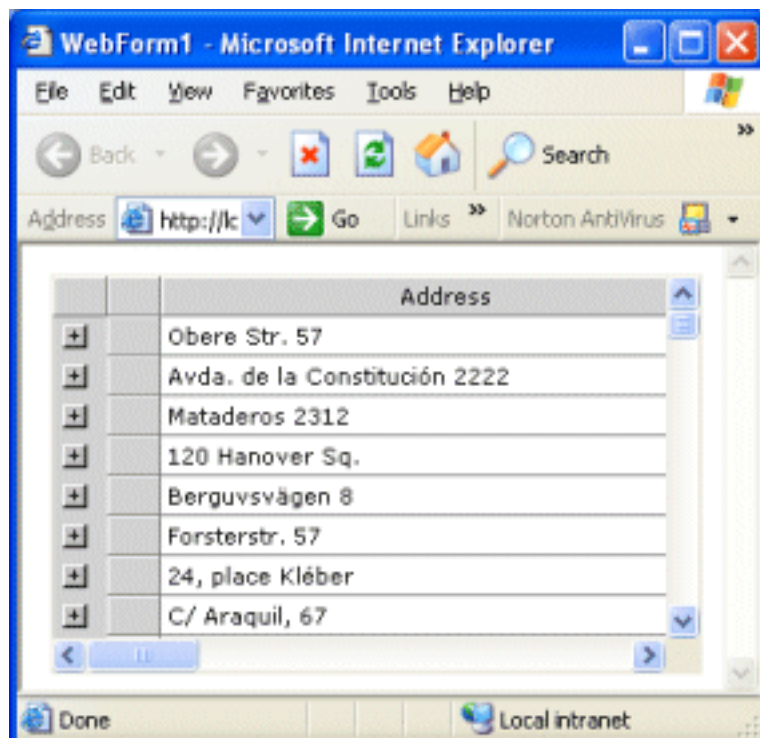
DataSource = DataSet11
DataMember = Customers
 - Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
' fill customers and orders data tables
OleDbDataAdapter1.Fill(DataSet11)
OleDbDataAdapter2.Fill(DataSet11)

' set grid view type and bind data
UltraWebGrid1.DisplayLayout.ViewType _
= Infragistics.WebUI.UltraWebGrid.ViewType.Hierarchical
UltraWebGrid1.DataBind()
```

- Run the project and after an initial delay you should see something like:

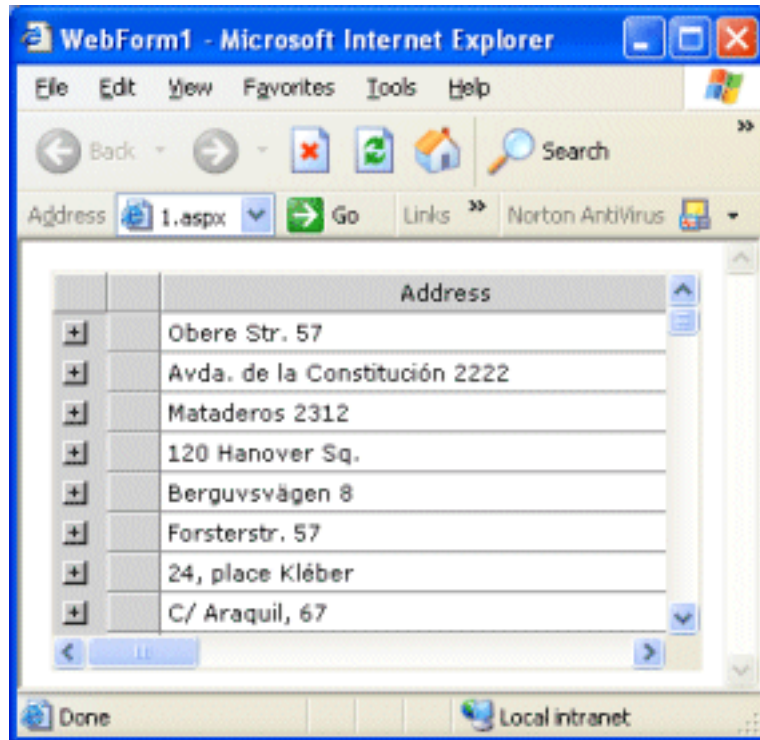


11. In the above example you can click the expansion indicators and the detail records display immediately with no post back, however the initial load time is excessive and the size of the web page source is in excess of two megabytes.
To solve this initial load time problem, add the following line of code before the UltraWebGrid1.DataBind() method invocation:

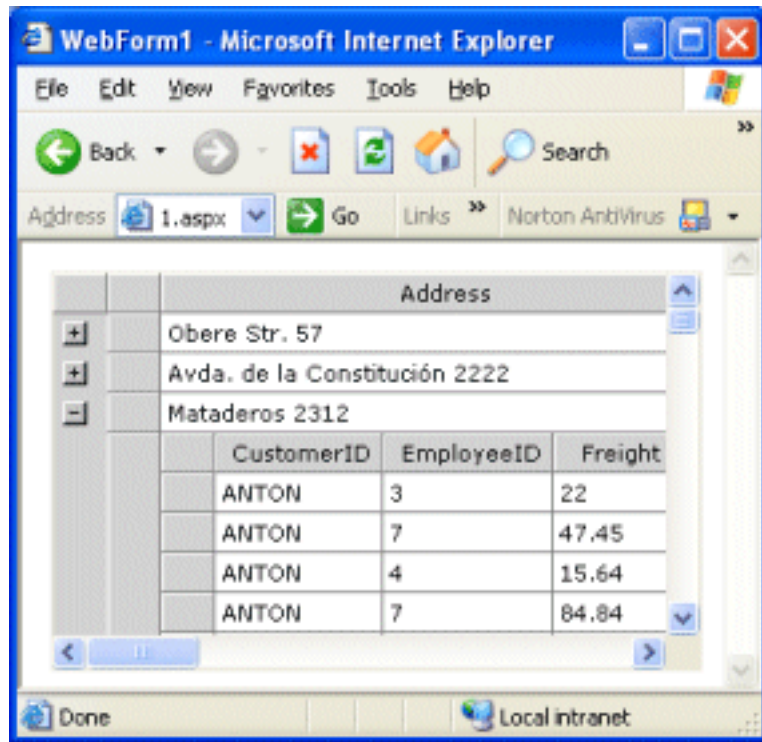
In Visual Basic:

```
UltraWebGrid1.DisplayLayout.LoadOnDemand _  
= Infragistics.WebUI.UltraWebGrid.LoadOnDemand.Automatic
```

12. Run the project and after a normal project start and page creation delay you will again see something like:



The difference is that none of the hierarchical band 1 row data was rendered and transmitted to the client. Now when you click on an expansion indicator you will see a post back to the server and the node will be populated with the appropriate rows:



Review

This tutorial shows how to use the automatic Load on Demand feature of the WebGrid and illustrates the resulting tremendous performance increase.

Multiple Data Key Fields

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

When processing user input from rows whose underlying data has multiple fields in the key, the row needs to maintain and transport these key field values. WebGrid version 2 introduces multiple key fields.

Questions

- I have underlying data that has two key fields. How do I tell WebGrid to transport both key field values?

Solution

Set the **DataKeyField** property of the band with the names of the key fields separated by commas, and when attempting to perform row operations, extract the key field values from the rows **DataKey** property as array elements such as:

```
strKeyValue1 = e.Cell.Row.DataKey(0).ToString
```

Sample Project

This sample project uses the Northind Order Details table as a source of data to demonstrate the WebGrid implementation of multiple key fields.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
6. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

7. Click on the WebGrid and set the following properties:

DataSource = DataSet11

DataMember = Customers

DisplayLayout.AllowUpdateDefault = Yes

DisplayLayout.CellClickActionDefault = Edit

8. Click on the Columns (Collection) property and open the Columns Collection Editor. Use the Up/Down arrows to arrange the column order as follows:

OrderID

ProductID

Quantity

UnitPrice

Discount

9. Click on OrderID and set **AllowUpdate** = No.
10. Click on ProductID and set **AllowUpdate** = No.
11. Tell the WebGrid there are two key fields by setting the DataKeyField property of the band to the names of the key fields separated by commas.
Click on the Bands (Collection) and open the Bands Collection Editor. Click on the Order Details band and set the following properties:

DataKeyField = "OrderID,ProductID"

12. From Toolbox, Web Forms; Add a TextBox control to the form designer. This text box will display the values of the key fields after the user changes a cell value.
13. Click on the TextBox control and set the following properties:

TextMode = MultiLine

14. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

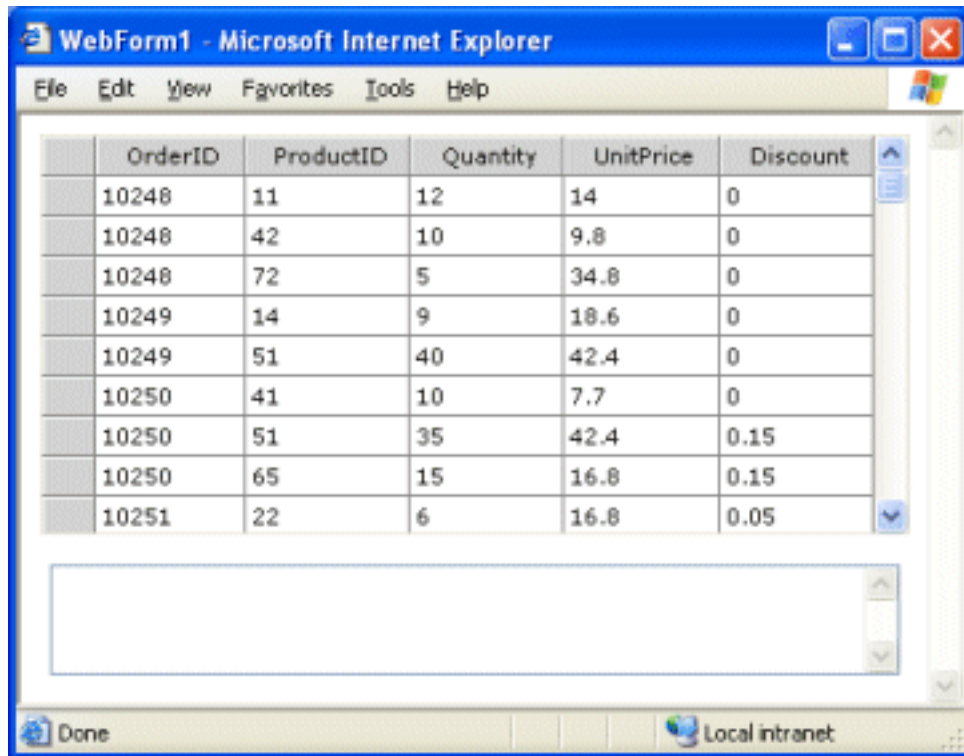
```
If Me.IsPostBack = False Then
    Me.OleDbSelectCommand1.CommandText _
    = "SELECT TOP 100 * FROM [Order Details]"
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

15. Add the following code to the WebGrid UpdateCell event:

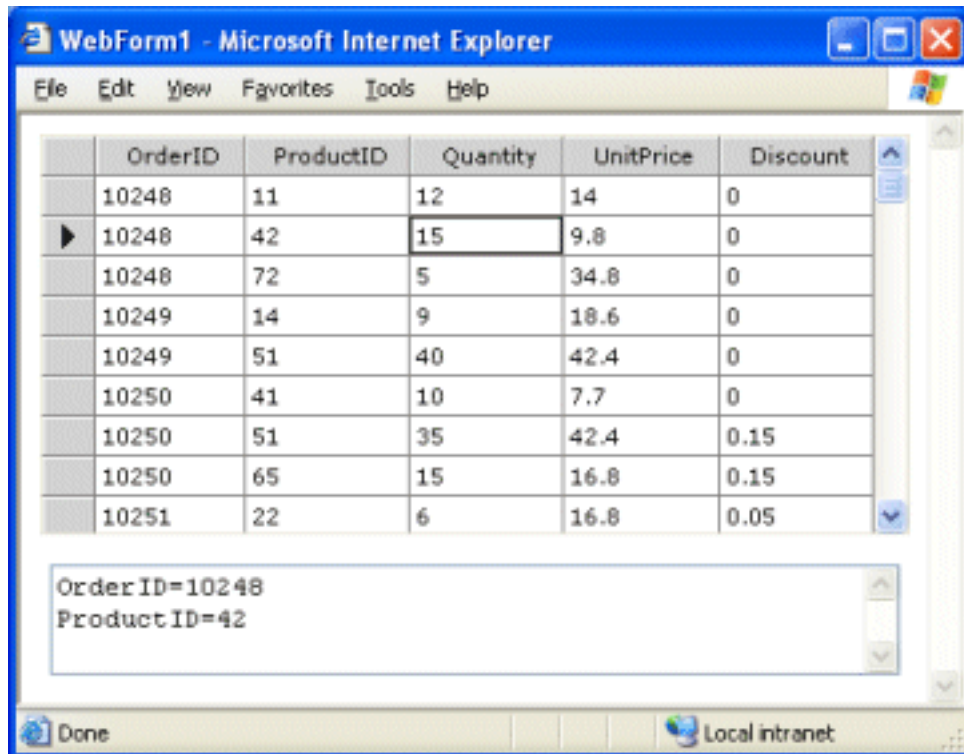
In Visual Basic:

```
' clear demonstration text box
TextBox1.Text = ""
' retrieve multiple key values
TextBox1.Text += "OrderID=" & _
& e.Cell.Row.DataKey(0).ToString & vbCrLf
TextBox1.Text += "ProductID=" & _
& e.Cell.Row.DataKey(1).ToString & vbCrLf
```

16. Run the project and you should see something like:



17. Click on a Quantity cell, change the value and press Tab or click on a different cell. WebGrid will perform a post back to the UpdateCell event which will populate the text box with the multiple key values and you should see something like:



Stop the project to end this tutorial.

Review

This tutorial shows how to specify multiple key fields by setting the Band DataKeyField property to the names of the key fields separated by commas, and retrieving the key field values from the Row DataKey property as array elements.

Client-Side Programming (Quick Start)

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

When the server side capabilities of an ASP.NET application or control do not provide enough control over actions on the client (web browser), the developer can insert client side jScript directly into the page HTML and perform operations on the client.

Questions

- How can I change the background color of cells in WebGrid that have been in edit mode?

Solution

Insert some jScript into the page HTML to react to the **AfterEnterEditMode** event and change the background color of the cell. Tell the WebGrid the name of the event handler in the `DisplayLayout.ClientSideEvents`.

AdterEnterEditModeHandler property.

Sample Project

Perform the following steps to connect the WebGrid to the Northwind customers table and add client side code to set the back color of changed cells.

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Customers and click Close, click ContactName, ContactTitle, Phone, CusotmerID and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to `Infragistics.WebUI.Shared`. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
6. Click the Infragistics.WebUI.Shared reference and set the following properties:
CopyLocal = True

- Click on the WebGrid and set the following properties:

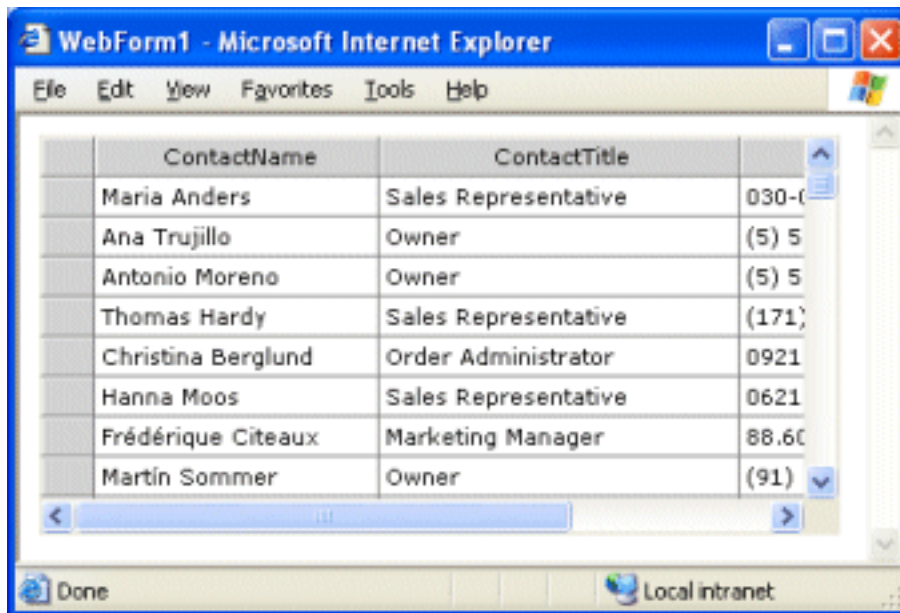
DataSource = DataSet11
DataMember = Customers

- Add the following code to the page's **Load** event to fill the dataset and bind it to the WebGrid:

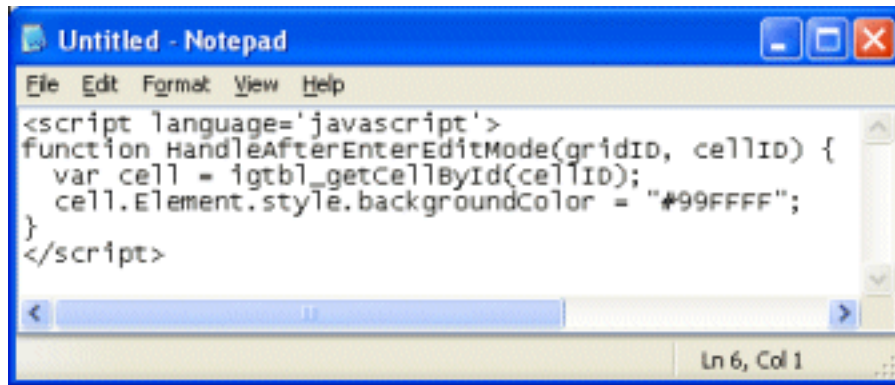
In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

- Run the project and depending on which version of the Northwind database you connect to you should see something like:



- Click around on the grid and notice the grid is not editable. Then stop the project.
- Now make the grid editable and set the cell click action to enter edit mode so you won't have to double-click to edit a cell. Also tell the grid which client-side event handler to use after the cell has been updated.
Right-click the grid and select Edit Layout. Set the following properties:
AllowUpdateDefault = Yes
CellClickActionDefault = Edit
ClientSideEvents.AfterEnterEditModeHandler = HandleAfterEnterEditMode
- Click the HTML tab at the bottom of the WebForm1.aspx designer to display the underlying HTML. Add the following JavaScript after the `</igtbl:UltraWebGrid>` tag and before the `</form>` tag:



```
File Edit Format View Help
<script language='javascript'>
function HandleAfterEnterEditMode(gridID, cellID) {
  var cell = igtbl_getCellById(cellID);
  cell.Element.style.backgroundColor = "#99FFFF";
}
</script>
Ln 6, Col 1
```

13. Run the project. Observe that as you click around on the cells and they enter edit mode. After you leave the cell the background color is a light blue. Stop the project when you are done.

Review

This tutorial shows how to add JavaScript code to the underlying HTML of the web form to respond to a client-side event and set the background color of potentially edited cells.

Client-Side Programming (Introduction)

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Application that deploy JavaScript into the client are typically difficult to debug, however, Visual Studio .NET has a very powerful JavaScript debugging environment that is comparable to server side Visual Basic and C# debugging.

Questions

- How do I setup Visual Studio .NET and Internet Explorer for client side debugging?

Solution

Perform the following steps to setup Visual Studio .NET (VS) and Internet Explorer (IE) for client side debugging:

1. In IE select Tools, Internet Options, Advanced, and turn OFF the option Disable Script Debugging so that it is unchecked.
2. Open the VS project that you wish to debug.
3. Press F5 to begin debugging.
4. Once the aspx page appears inside IE, switch back to the debugger.
5. Click on the Debug menu and select Windows, Running Documents From the list of displayed running documents, select and open the file containing the JavaScript you wish to debug.
6. In the JavaScript file go to the function(s) that you wish to debug and set breakpoints on them.
7. Switch back to the browser and perform actions that invoke the breakpoints. The debugger will break at the lines of code you have marked.

Within a break point you can look at the call-stack and watch windows that contain all of the variables and objects that are currently in scope within the JavaScript environment as well as the entire HTML DOM.

To see initialization code execute during page initialization, hit F5 to refresh the page after setting breakpoints.

Sample Project

Perform the following steps to create a new JavaScript project, setup Visual Studio .NET and Internet Explorer for client side debugging, and perform debugging tasks:

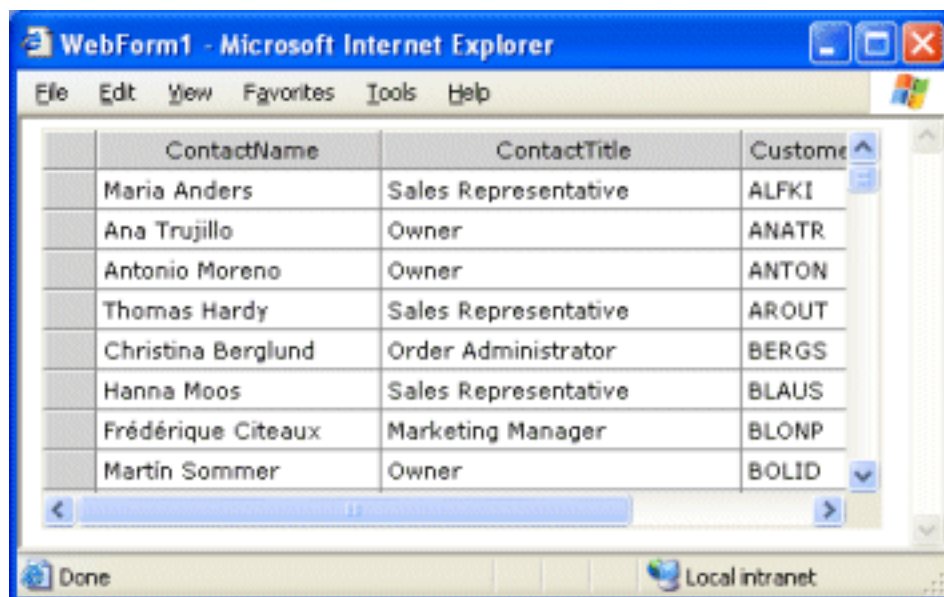
1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`

- Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Customers and click Close, click ContactName, ContactTitle, Phone, CusotmerID and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK.
 4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
 5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
 6. Click the Infragistics.WebUI.Shared reference and set the following properties:
CopyLocal = True
 7. Click on the WebGrid and set the following properties:
DataSource = DataSet11
DataMember = Customers
 8. Add the following code to the page's **Load** event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

9. Run the project and depending on which version of the Northwind database you connect to you should see something like:



Click around on the grid and notice the grid is not editable. Then stop the project.

10. Now make the grid editable and set the cell click action to enter edit mode so you don't have to double click to edit a cell. Also tell the grid which client-side event handler to use after the cell has been updated.
Right-click the grid and select Edit Layout. Set the following properties:

AllowUpdateDefault = Yes

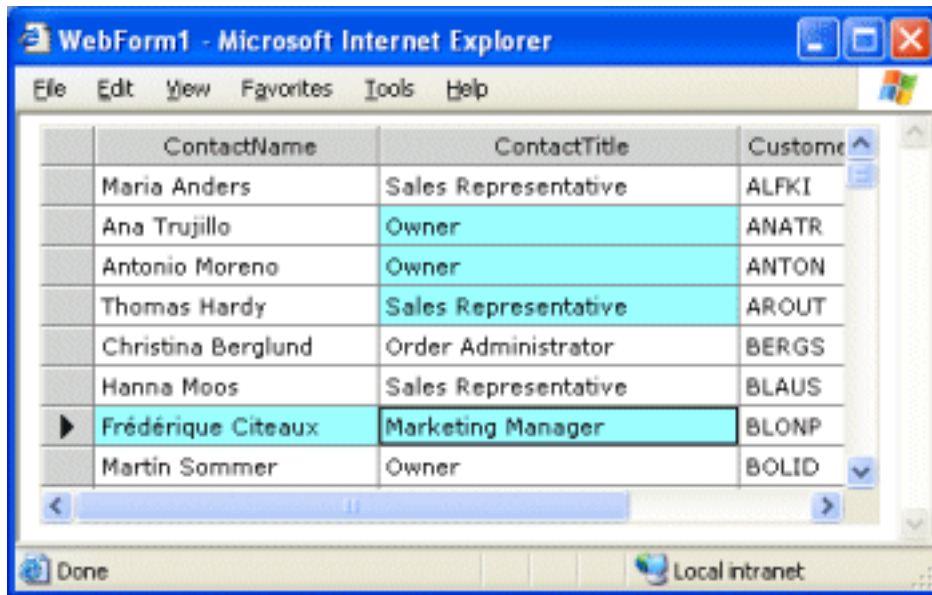
CellClickActionDefault = Edit

ClientSideEvents.AfterEnterEditModeHandler = HandleAfterEnterEditMode

11. Click the HTML tab at the bottom of the WebForm1.aspx designer to display the underlying HTML. Add the following JavaScript after the </igtbl:UltraWebGrid> tag and before the </form> tag:

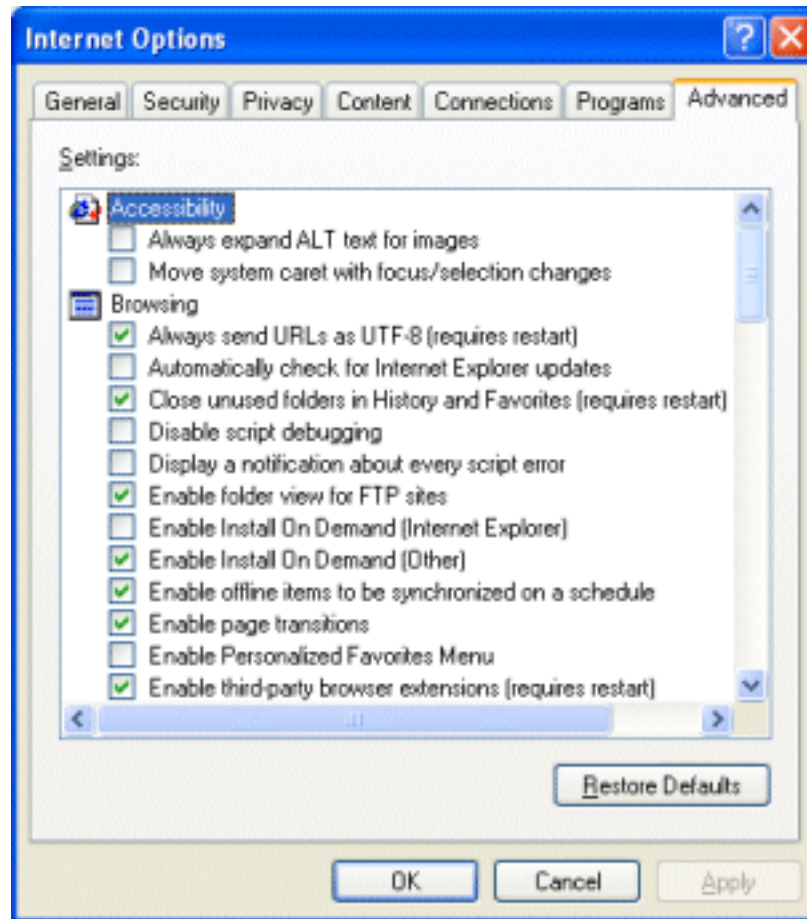
```
<script language='javascript'>
function HandleAfterEnterEditMode(gridID, cellID) {
    var cell = igtbl_getCellById(cellID);
    cell.Element.style.backgroundColor = "#99FFFF";
}
</script>
```

12. Run the project. Observe that as you click around on the cells and they enter edit mode. After you leave the cell the background color is a light blue:

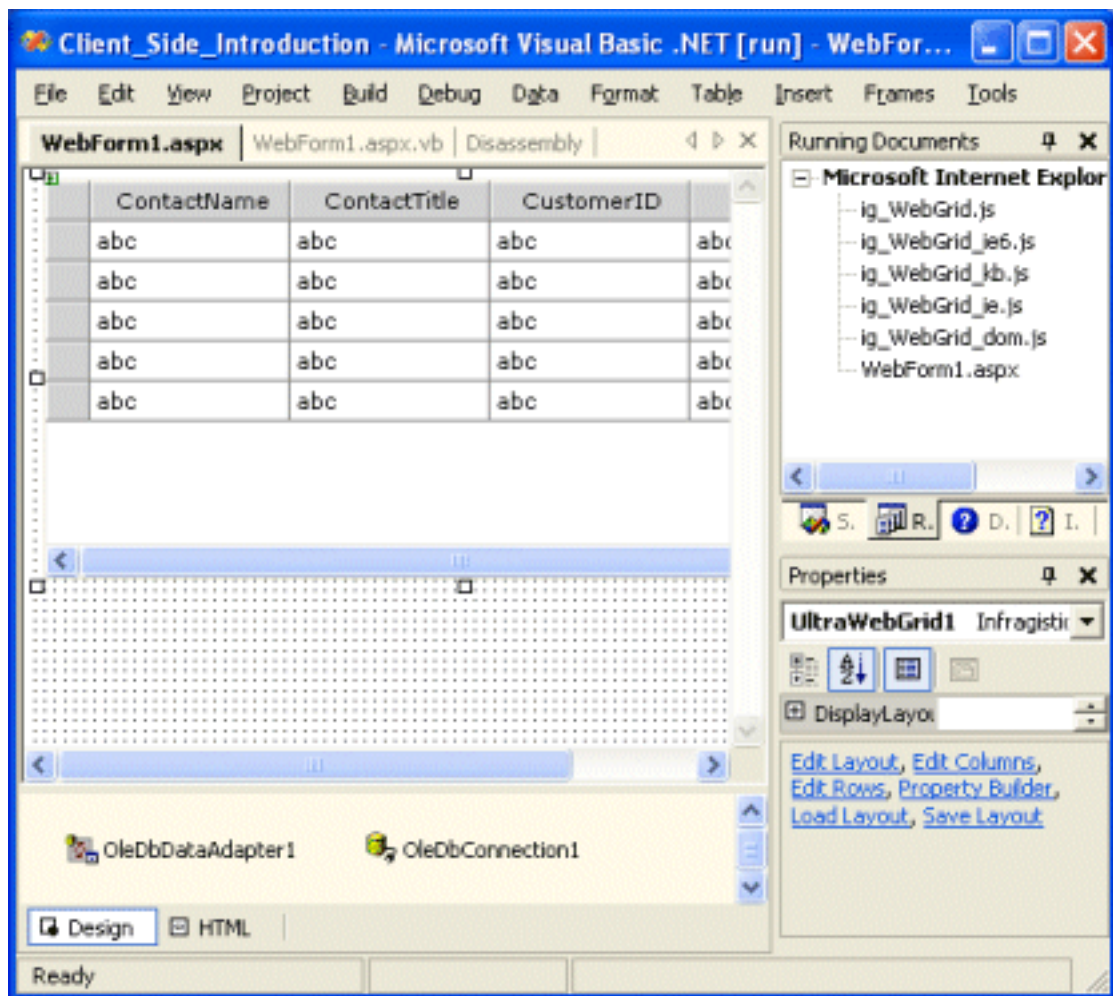


Stop the project.

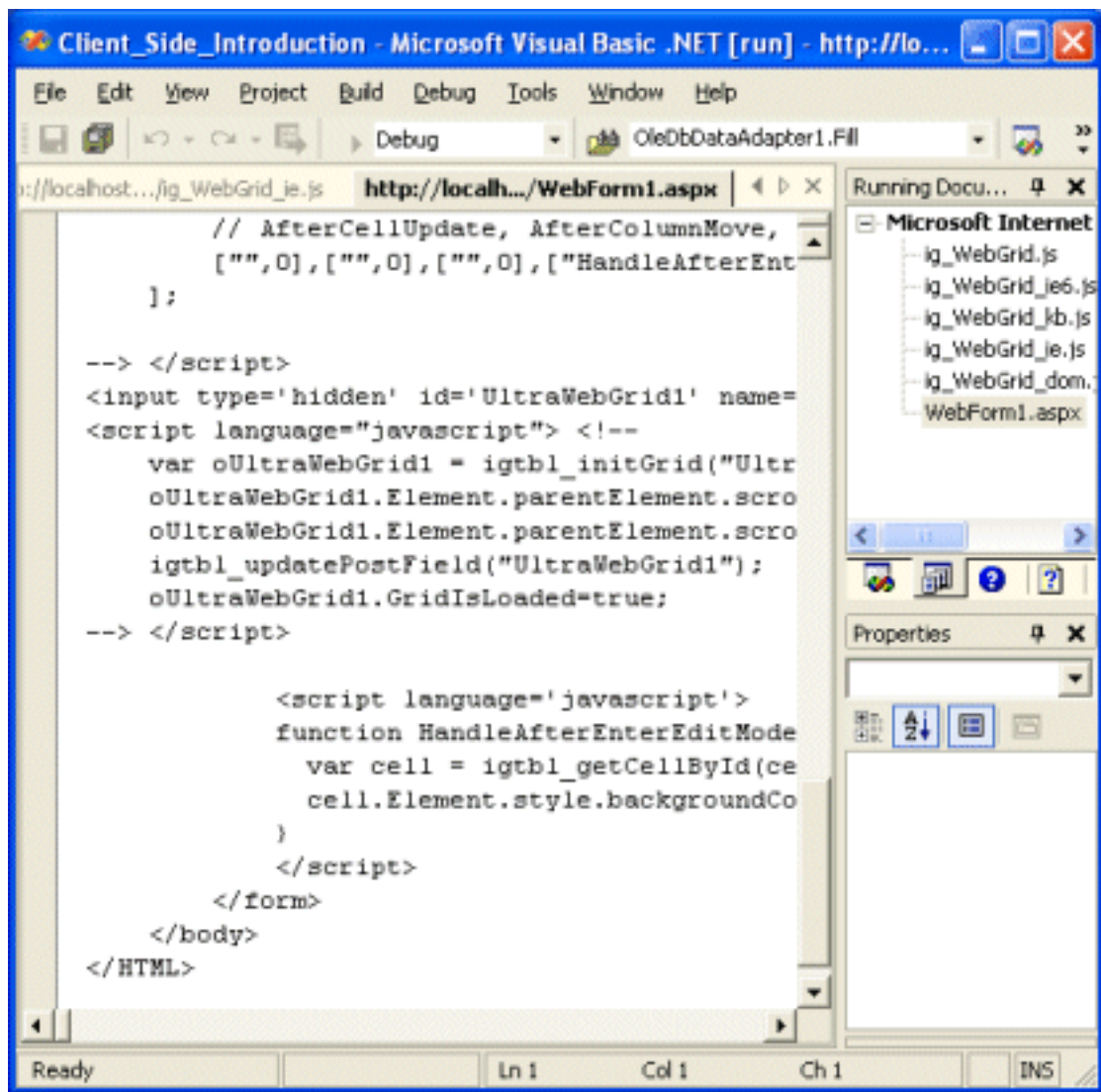
13. Perform the following steps to make sure Internet Explorer is configured to perform script debugging:
 - Launch Internet Explorer.
 - From the menu select Tools, Internet Options, Advanced and make sure the property Disable script debugging is NOT checked:



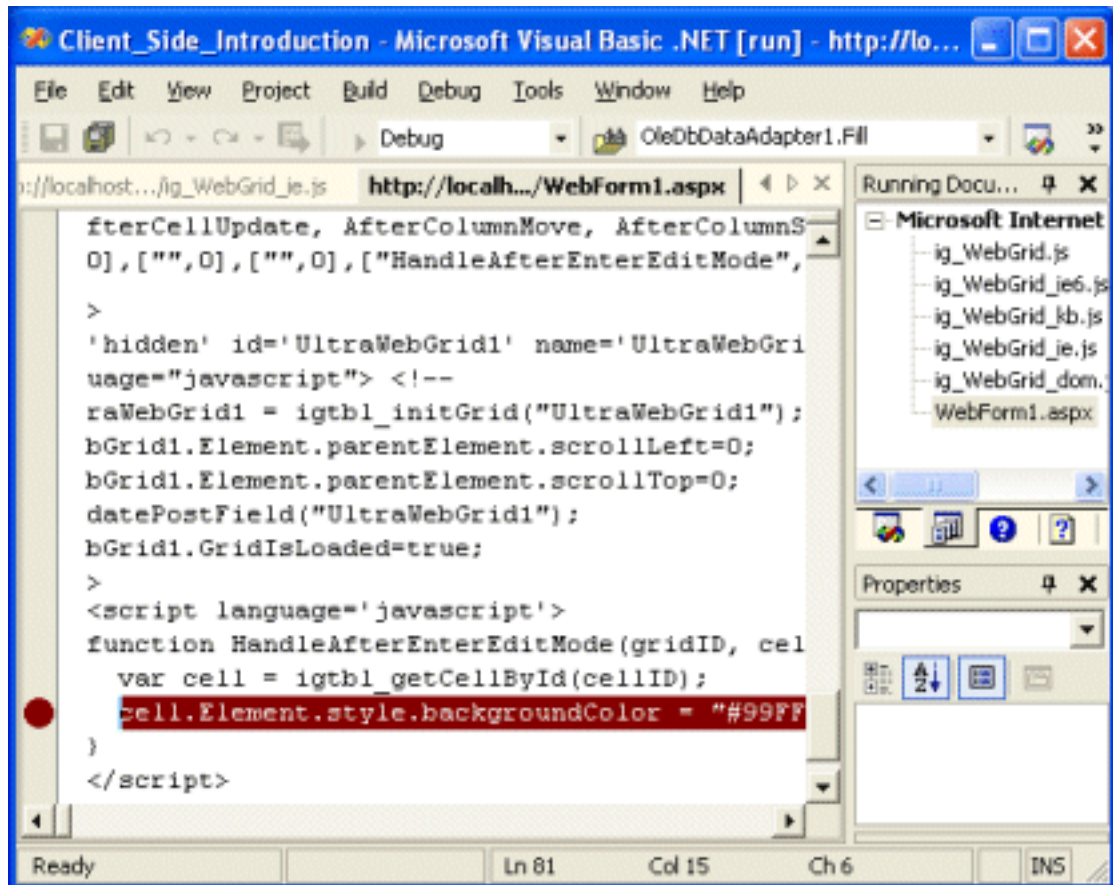
14. Press F5 to begin debugging.
15. When the aspx page appears inside IE, switch back to the debugger.
16. From the Visual Studio main menu select Debug, Windows, Running Documents and you should see something like:



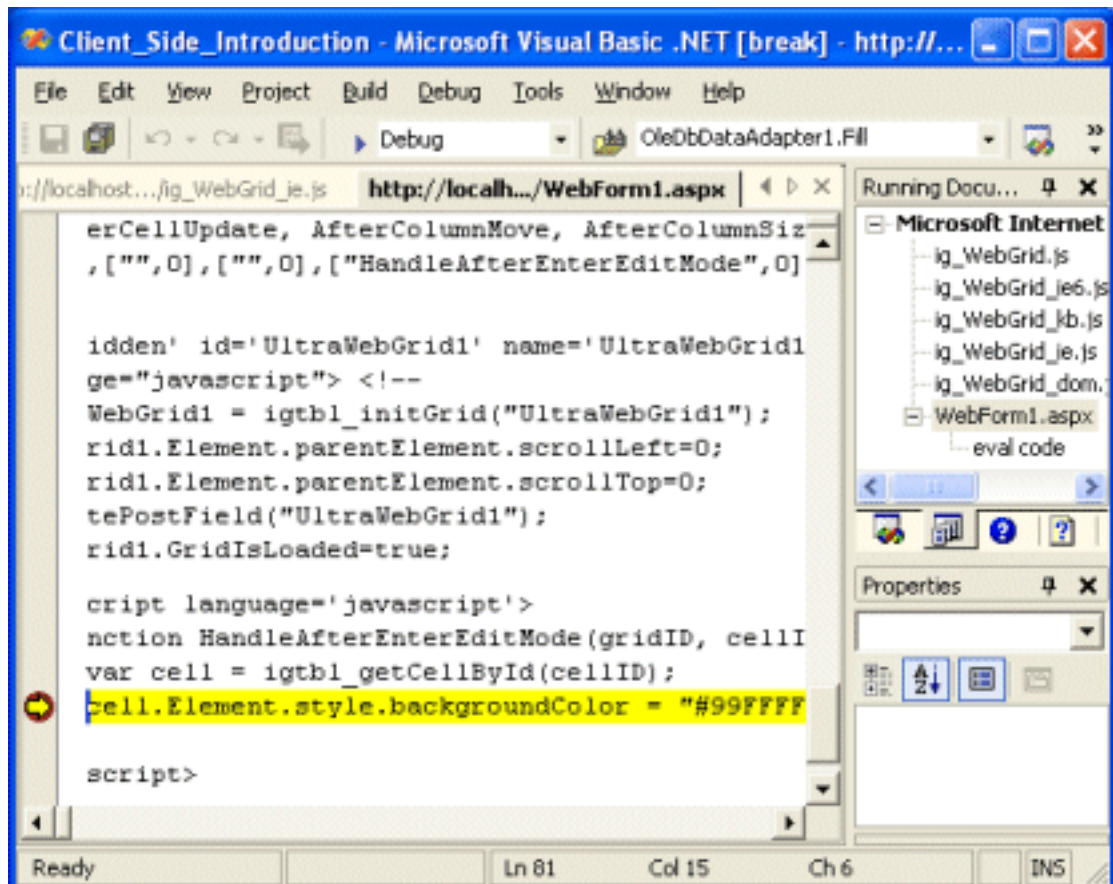
17. Observe the Running Documents dialog. It shows all of the .js files and .aspx files running on the client. The JavaScript you added is in WebForm1.aspx. Double-click the WebForm1.aspx node to display the rendered HTML in the browser and scroll to the bottom. You should see something like:



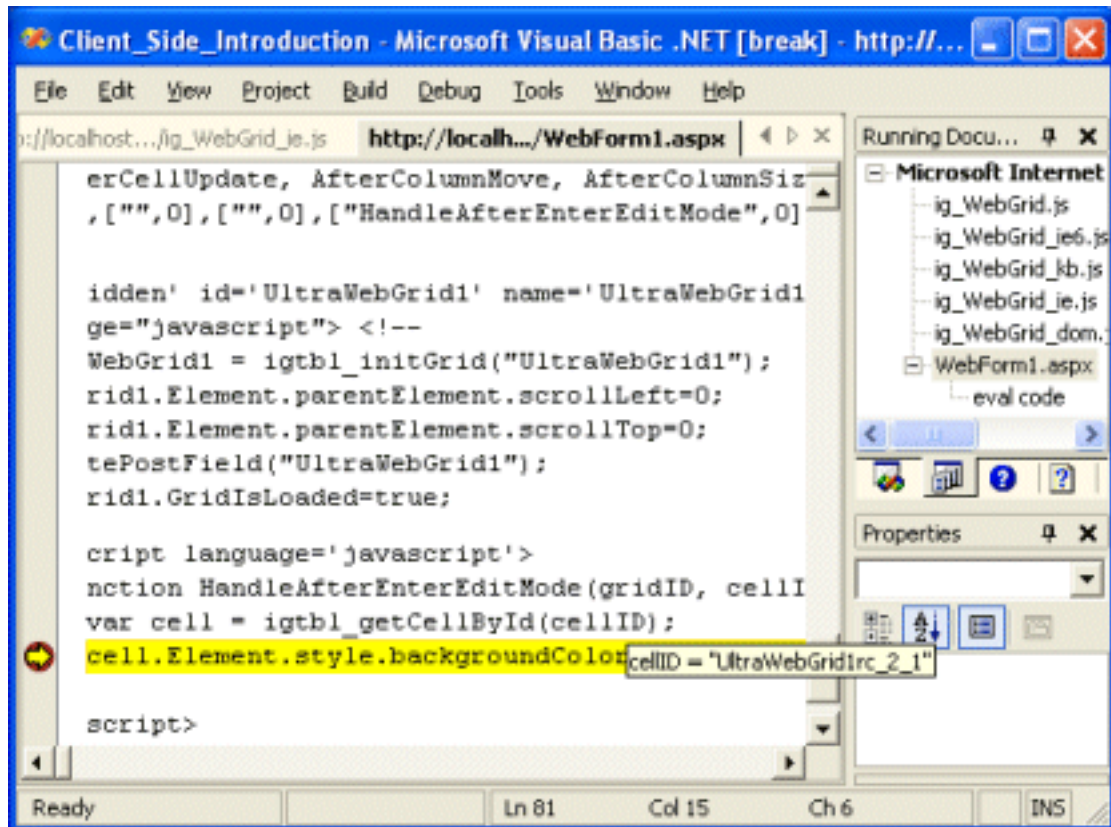
18. Notice the function `HandleAfterEnterEditMode()` displays. Set a break point on the line of code starting with `cell.Element.style.backgroundColor` by clicking in the left margin of the Visual Studio code editor and you should see something like:



19. Select the browser displaying the WebGrid and perform actions that invoke the line of code containing the breakpoint. Visual Studio will stop before the selected line of code executes and display something like:



20. Observe the line highlighted in yellow is the line containing the break point. Move the mouse over the cellID variable in the previous line and you should observe the value of this variable to display in a tool tip:



Click Continue and the program continues. Then stop the project.

21. Within a break point you can look at the call-stack and watch windows that contain all of the variables and objects that are currently in scope within the JavaScript environment as well as the entire HTML DOM.
To see initialization code execute during page initialization, hit F5 to refresh the page after setting breakpoints.

Review

This tutorial shows the developer how to create a simple client side script, configure Internet Explorer to work in debug mode, and use Visual Studio to set break points in the client side code and view client side variable values.

Client-Side Programming (Intermediate)

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

There are financial applications where the summation of rows and columns is required. Postbacks to the server to perform these calculations make the thin-client deployment cumbersome to use and limited bandwidth aggravates the problem.

The solution is to perform the summations and other calculations on the client as when the user exits a field.

Questions

- How can I make WebGrid work more like a spreadsheet and provide row and column summations?

Solution

Perform the calculations and add the calculated information to the grid dynamically on the client-side using JavaScript .

Sample Project

This sample application presents a number of rows and columns which are program generated and through the use of client-side jScript the row and column totals are calculated.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
3. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
4. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

5. Before getting started with the client-side code, clean this grid up a little bit by formatting the quarterly value columns and making the ID, Name and Total columns not editable. To do this, create a strongly-typed data set:
 - a. From the main menu, select Project, Add New Item, DataSet and change the name to `Accounts.xsd`. Select Open.
 - b. Right-click the Accounts.xsd designer and select Add, New Element. An element should display.
 - c. Change the name from element1 to "Accounts".
 - d. Add the following elements:

◆ E	Accounts	(Accounts)
🔑 E	Id	string
E	Name	string
E	Q1	decimal
E	Q2	decimal
E	Q3	decimal
E	Q4	decimal
E	Total	decimal

6. Select the WebForm1.aspx designer.
7. From the Toolbox, Data tab, drag a DataSet onto the designer form and click OK.
8. Click the web grid and set the following properties in the properties dialog:
 - DataSource** = Accounts1
 - DataMember** = Accounts
 - DisplayLayout.AllowUpdateDefault** = Yes
 - DisplayLayout.CellClickActionDefault** = Edit
 - DisplayLayout.ColFootersVisibleDefault** = Yes
9. Now the grid knows what our data looks like, click the Columns (collection) and open the Columns Collection Editor and set the following properties for each column:

Id:

AllowUpdate = No

Name:

AllowUpdate = No

Q1:

Format = Currency

CellStyle.HorizontalAlign = Right

FooterTotal = Sum

FooterStyle.HorizontalAlign = Right

Q2:

Format = Currency

CellStyle.HorizontalAlign = Right

FooterTotal = Sum

FooterStyle.HorizontalAlign = Right

Q3:

Format = Currency

CellStyle.HorizontalAlign = Right

FooterTotal = Sum

FooterStyle.HorizontalAlign = Right

Q4:

Format = Currency
CellStyle.HorizontalAlign = Right
FooterTotal = Sum
FooterStyle.HorizontalAlign = Right

Total:

AllowUpdate = No
Format = Currency
CellStyle.HorizontalAlign = Right
CellStyle.Font.Bold = True

10. Add the following private method to WebForm1.aspx to create some sample financial data:

In Visual Basic:

```
Private Sub MakeAccountsDataTable(ByRef v_ds As DataSet)
    ' declare quarter names
    Dim QuarterName() As String = {"Q1", "Q2", "Q3", "Q4"}

    ' declare a DataTable to contain the program generated data
    Dim dataTable As DataTable = v_ds.Tables("Accounts")

    ' add some rows
    Dim row As DataRow
    Dim intPtr As Integer
    Dim intAccount As Integer
    Dim intQtr As Integer

    For intPtr = 1 To 9
        row = dataTable.NewRow()
        intAccount = intPtr * 100
        row("ID") = intAccount
        row("Name") = "Account " & intAccount.ToString

        ' put some random data into the value columns
        For intQtr = 1 To 4
            row(QuarterName(intQtr - 1)) _
                = Math.Round((Rnd() * 1000 - 100), 2)
        Next
        dataTable.Rows.Add(row)
    Next
End Sub
```

11. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.MakeAccountsDataTable(Me.Accounts1)
    Me.UltraWebGrid1.DataBind()
End If
```

12. Run the project and you should see something like:

WebForm1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

	Id	Name	Q1	Q2	Q3	Q4	Total
	100	Account 100	\$605.55	\$433.42	\$479.52	\$189.56	
	200	Account 200	\$201.95	\$674.74	-\$85.98	\$660.72	
	300	Account 300	\$714.49	\$609.04	-\$54.65	\$314.03	
	400	Account 400	\$762.62	\$690.48	\$273.54	\$861.95	
	500	Account 500	\$771.45	-\$43.76	\$849.56	\$264.02	
	600	Account 600	\$424.87	\$667.11	-\$46.50	\$492.46	
	700	Account 700	\$368.70	\$198.17	\$522.70	\$547.82	
	800	Account 800	\$163.79	\$179.34	\$729.80	\$724.60	
	900	Account 900	\$489.16	\$886.09	\$810.96	\$126.87	
			\$4,502.58	\$4,294.63	\$3,478.95	\$4,182.03	

Done Local intranet

Stop the project.

Notice the Total column is empty. Assuming you do not retrieve these totals from the database, you must add some server-side code to seed these values.

13. In the WebGrid's **InitializeRow** event add the following code to calculate the row totals:

In Visual Basic:

```
' calculate total column value for each row
e.Row.Cells.FromKey("Total").Value = _
e.Row.Cells.FromKey("Q1").Value _
+ e.Row.Cells.FromKey("Q2").Value _
+ e.Row.Cells.FromKey("Q3").Value _
+ e.Row.Cells.FromKey("Q4").Value
```

14. Run the project and you should see something like:

WebForm1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

	Id	Name	Q1	Q2	Q3	Q4	Total
	100	Account 100	\$605.55	\$433.42	\$479.52	\$189.56	\$1,708.05
	200	Account 200	\$201.95	\$674.74	-\$85.98	\$660.72	\$1,451.43
	300	Account 300	\$714.49	\$609.04	-\$54.65	\$314.03	\$1,582.91
	400	Account 400	\$762.62	\$690.48	\$273.54	\$861.95	\$2,588.59
	500	Account 500	\$771.45	-\$43.76	\$849.56	\$264.02	\$1,841.27
	600	Account 600	\$424.87	\$667.11	-\$46.50	\$492.46	\$1,537.94
	700	Account 700	\$368.70	\$198.17	\$522.70	\$547.82	\$1,637.39
	800	Account 800	\$163.79	\$179.34	\$729.80	\$724.60	\$1,797.53
	900	Account 900	\$489.16	\$886.09	\$810.96	\$126.87	\$2,313.08
			\$4,502.58	\$4,294.63	\$3,478.95	\$4,182.03	

Done Local intranet

Stop the project.

Now you have seeded the grid with column values and totals. The challenge now is to insert client-side JavaScript to keep the Total column and Footer values updated as the user changes quarterly totals.

15. First take a look at keeping the row total updated. You can setup client-side code that handles the **AfterExitEditMode** event. From the WebForm1.aspx designer, click the web grid and set the following property:

In Visual Basic:

```
ClientSideEvents.AfterExitEditModeHandler = HandleExitEditMode
```

16. Click the HTML tab at the bottom of the WebForm1.aspx designer to display the underlying HTML. Add the following JavaScript after the `</igtbl:UltraWebGrid>` tag and before the `</form>` tag:

```
<script language='javascript'>

function HandleExitEditMode(gridID, cellID) {
    /* ACCUMULATE VALUE FOR THIS ROW */

    // get reference to row object
    var row = igtbl_getRowById(cellID);

    // retrieve Q1 value
    var rowTotal = row.getCellFromKey("Q1").getValue();

    // add values of Q2, Q3, and Q4
    rowTotal += row.getCellFromKey("Q2").getValue();
    rowTotal += row.getCellFromKey("Q3").getValue();
    rowTotal += row.getCellFromKey("Q4").getValue();

    // round to two decimal positions
    rowTotal = Math.round(rowTotal * 100) / 100

    // place value in Total cell for this row
    row.getCellFromKey("Total").setValue(rowTotal.toString());
}
</script>
```

17. Run the project, go to the third row and set the quarterly values to 100, 200, 300 and 400, and you should see something like:

Id	Name	Q1	Q2	Q3	Q4	Total
100	Account 100	\$605.55	\$433.42	\$479.52	\$189.56	\$1,708.05
200	Account 200	\$201.95	\$674.74	-\$85.98	\$660.72	\$1,451.43
300	Account 300	\$100.00	\$200.00	\$300.00	\$400.00	\$1,000.00
400	Account 400	\$762.62	\$690.48	\$273.54	\$861.95	\$2,588.59
500	Account 500	\$771.45	-\$43.76	\$849.56	\$264.02	\$1,841.27
600	Account 600	\$424.87	\$667.11	-\$46.50	\$492.46	\$1,537.94
700	Account 700	\$368.70	\$198.17	\$522.70	\$547.82	\$1,637.39
800	Account 800	\$163.79	\$179.34	\$729.80	\$724.60	\$1,797.53
900	Account 900	\$489.16	\$886.09	\$810.96	\$126.87	\$2,313.08
		\$3,888.09	\$3,885.59	\$3,833.60	\$4,268.00	

Notice the row total is updated as the cells are exited. Then stop the project.

- To update the totals in the column footers, add the following JavaScript code to the **HandleExitEditMode** event handler:

```

/* ACCUMULATE VALUE FOR THIS COLUMN */

// get reference to the rows collection
var rows = igtbl_getGridById(gridID).Rows;

// get reference to column
var cell = igtbl_getCellById(cellID);
var column = cell.Column;

// see if there are rows to process
var colTotal = 0;
if (rows.length >= 1.0)
{
    // add column value of each row
    for (var i = 0; i < rows.length; i++)
    {
        colTotal += rows.getRow(i).getCellByColumn(column).getValue();
    }

    // place value in column footer
    var cell = igtbl_getElementById(cellID);
    if (cell.parentNode.parentNode.nextSibling)
    {
        var footer=cell.parentNode.parentNode.nextSibling.childNodes[0].childNodes
        [cell.cellIndex];
        footer.innerHTML = igtbl_Mask(gridID, colTotal.toString(), 14, "$###,###,
        ##0.00");
    }
}

```

- Run the project and make all of the column values in one of the columns equal to \$100.00 and you should see something like:

	Id	Name	Q1	Q2	Q3	Q4	Total
	100	Account 100	\$605.55	\$100.00	\$479.52	\$189.56	\$1,374.63
	200	Account 200	\$201.95	\$100.00	-\$85.98	\$660.72	\$876.69
	300	Account 300	\$100.00	\$100.00	\$300.00	\$400.00	\$900.00
	400	Account 400	\$762.62	\$100.00	\$273.54	\$861.95	\$1,998.11
	500	Account 500	\$771.45	\$100.00	\$849.56	\$264.02	\$1,985.03
	600	Account 600	\$424.87	\$100.00	-\$46.50	\$492.46	\$970.83
	700	Account 700	\$368.70	\$100.00	\$522.70	\$547.82	\$1,539.22
	800	Account 800	\$163.79	\$100.00	\$729.80	\$724.60	\$1,718.19
▶	900	Account 900	\$489.16	\$100.00	\$810.96	\$126.87	\$1,526.99
			\$3,888.09	\$900.00	\$3,833.60	\$4,268.00	

Stop the project and notice the total of the column with \$100.00 values is now \$900.00 as expected.

Review

This project shows how to perform intermediate level client-side jScript operations on rows, columns and footers.

Client-Side Programming (Advanced)

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

When working with WebGrid Row Templates, the default text box created for data entry may not be adequate to provide user input validation, and/or the application might require the use of Check Boxes or Radio Buttons. In these cases, the developer will need to write some client side jScript to translate between the WebGrid cell values the control values on the template.

Questions

- I have a column on my grid which can contain only one of two values WA for Washington, and IN for International. I want to use RadioButtons on my template for user input. How can I translate from the grid cell values to the RadioButton values on the template?

Solution

Write some client side JavaScript to perform this translation.

Sample Project

Perform the following steps to connect the Access Northwind Employees database to an Infragistics WebGrid:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Employees and click Close, click EmployeeID, FirstName, LastName, and Region and click OK.
 - Click Finish
3. From the main Menu select Data, Generate Dataset and click OK
4. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
5. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
6. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

7. Click on the WebGrid and set the following properties:

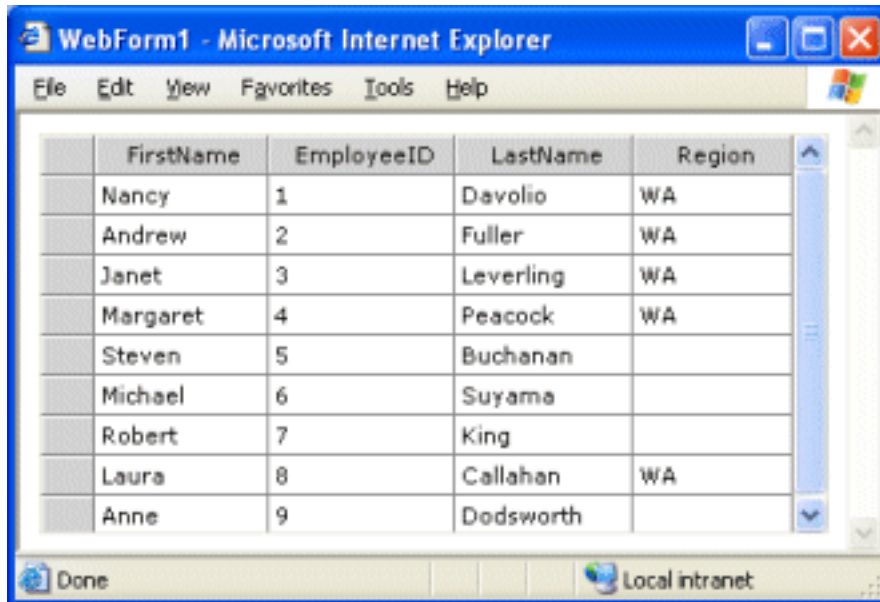
DataSource = DataSet11 **DataMember** = Employees

8. Add the following code to the page load event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.UltraWebGrid1.DataBind()
End If
```

9. Run the project and depending on which version of the Northwind database you connect to you should see something like:



Stop the project.

10. Arrange the columns and clean up the grid format:

- Click the WebGrid on the WebForm1.aspx designer and from the properties dialog select Columns (Collection) to open the Columns Collection Editor.
- Use the Up/Down arrows to arrange the columns as follows:

EmployeeID
FirstName
LastName
Region

- Set the following properties of the columns:

EmployeeID:

HeaderText = ID
Width = 30px

FirstName:

HeaderText = First Name

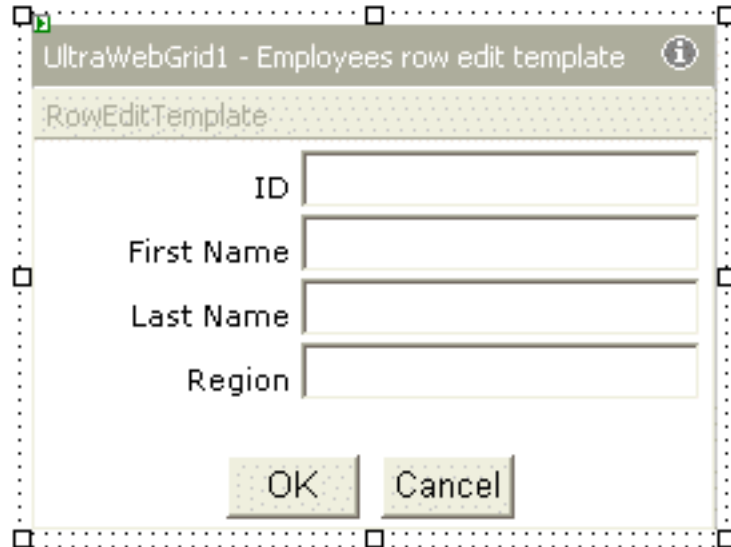
LastName:

HeaderText = Last Name

11. Right-click the WebGrid and select Edit Layout. Set the following properties in the Edit DisplayLayout dialog:

AllowUpdateDefault = RowTemplateOnly

12. Right-click the WebGrid and select Edit Template, Employees row edit template and you should see something like:



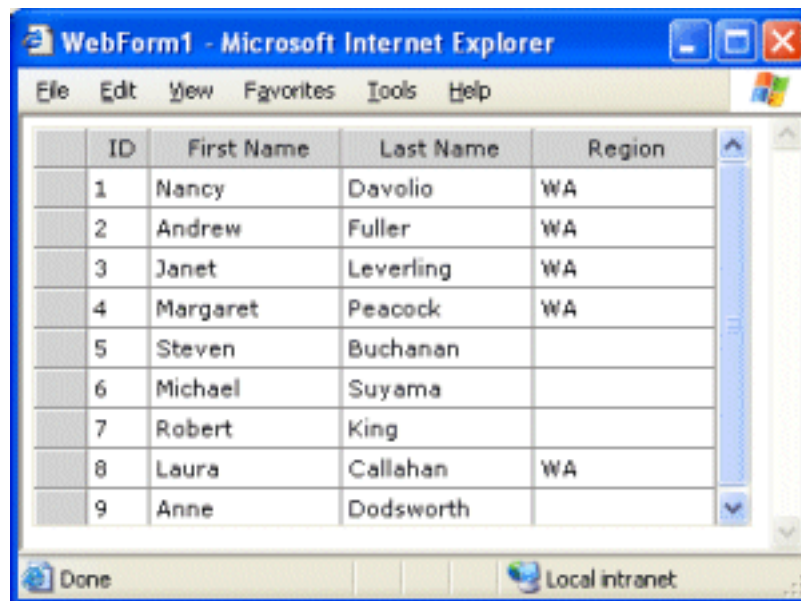
Observe the four columns from the grid row are editable with text boxes.

13. Reduce the width of the ID text box and set the following properties:

readonly = True

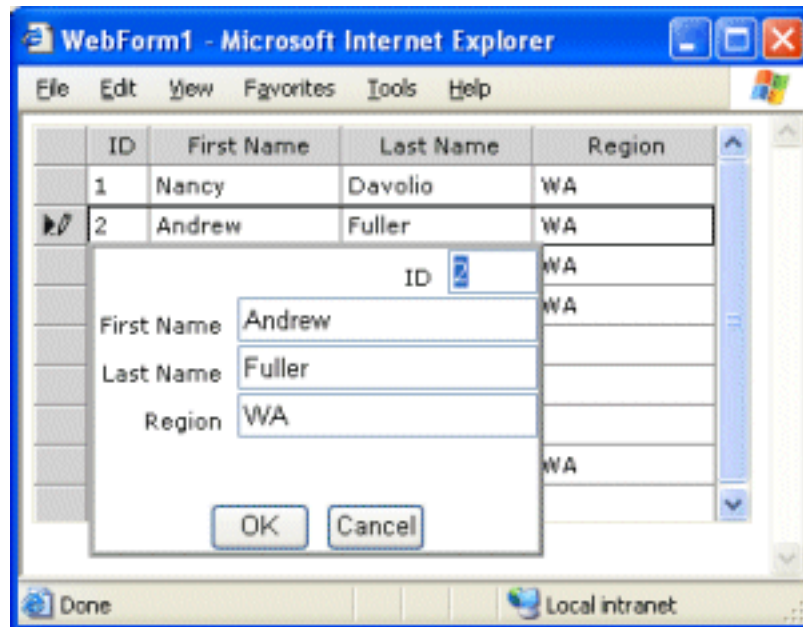
14. Right-click the template form and select End Template Editing.

15. Run the project and you should see something like:



ID	First Name	Last Name	Region
1	Nancy	Devolio	WA
2	Andrew	Fuller	WA
3	Janet	Leverling	WA
4	Margaret	Peacock	WA
5	Steven	Buchanan	
6	Michael	Suyama	
7	Robert	King	
8	Laura	Callahan	WA
9	Anne	Dodsworth	

16. Click on the row selector for row two, then click it again and you should see something like:



Notice the Region text field shows. Stop the project when you are through.

- For this application you will use radio buttons for two regions. Button 1 is for WA or Washington, and Button2 is for IN for International. To make this work you will need to translate between the cell values and the radio button state.

Note Before continuing to add the radio buttons, there a number of issues you need to be aware of.

First, you will notice the controls in the row template are standard HTML controls, not WebForm controls. First, there is no need to use WebForm controls because they are never going to interface with the server. Second, rendered WebForm controls have a difficult time sending their Control ID to the client-side events. So using WebForm controls for row templates is not a good idea in this instance.

Second, placing HTML controls directly on the row template designer can be problematic due to a behavioral issue in Visual Studio. The behavior causes the ID values of the text boxes to be reset to default values when a radio button is added to the form. (This issue may affect you whenever you drag and drop a HTML control onto the row template designer. You will have to determine if you are affected on a case-by-case basis.) This behavior poses a serious problem, because for the text boxes that automatically bind to the underlying row cells, the ID is in a special format and must not be changed. Because Visual Studio changes these ID values, the row template will no longer work properly.

To address both of these issues, it is suggested that you create a separate web form, set the layout to "flow layout" and create your row template layout using this form. Then copy and paste the HTML into the designer of the form where the row template actually resides.

From the main menu, select Project, Add Web Form In the Add New Item dialog, change the name to Designer.aspx and click Open. Right click the Designer.aspx form and select Properties. On the DOCUMENT Property Pages, change the Page Layout: to FlowLayout.

- Use the Designer.aspx form to layout the Region radio buttons:
 - For radio buttons to work, the need to be placed on a panel. From the Toolbox, HTML tab, drag and drop a flow layout panel onto the designer. Set the width to about 240px and the height to about 15px.
 - Activate the layout panel by clicking on the panel and clicking again. Type Region.
 - From the Toolbox, HTML tab, drag and drop a radio button onto the panel. Set the ID property of this radio button to rbWashington.
 - Click to the right of the radio button and type Washington.

- From the Toolbox, HTML tab, drag and drop another radio button. Set the ID property of this radio button to rbInternational.
- Click to the right of the radio button and type International.

19. The HTML underlying this form now contains the HTML code you need to replace the existing region text box on the row template. To retrieve this HTML:

- Click the HTML tab at the bottom of the designer.
- Inside the form tag you will find the HTML we are looking for. Mark this HTML text and copy it to the clipboard:

```
<DIV style="WIDTH: 240px; HEIGHT: 25px" ms_positioning="FlowLayout">
```

```
Region
```

```
<INPUT id="rbWashington" type="radio" value="rbWashington" name="RadioGroup">
```

```
Washington
```

```
<INPUT id="rbInternational" type="radio" value="rbInternational" name="RadioGroup">
```

```
International
```

```
</DIV>
```

20. Replace the existing Region text and text box with HTML on the clipboard:

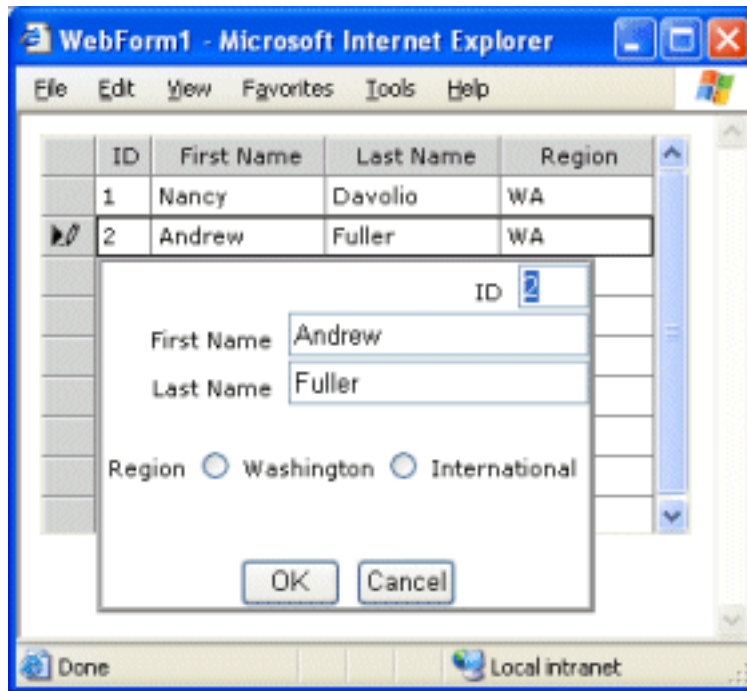
- Select the WebForm1.aspx designer and click the HTML tab at the bottom of the designer.
- Locate the Region text and text box, select the text, and click Delete.
- Paste the text from the clipboard.

21. Click the Design tab at the bottom of the form and display the row template. You should see something like:

The screenshot shows a dialog box titled "UltraWebGrid1 - Employees row edit template". Inside the dialog, there is a "RowEditTemplate" section. It contains the following elements:

- An "ID" label followed by a text input field.
- A "First Name" label followed by a text input field.
- A "Last Name" label followed by a text input field.
- A "Region" label followed by two radio buttons: "Washington" and "International".
- At the bottom, there are "OK" and "Cancel" buttons.

22. You now have the two radio buttons on the on the row template. Run the project, click on a row selector, then click it again and you should see something like:



Notice both the Washington and International radio buttons display. Click on the radio buttons and notice that only one button can be selected at any one time. Stop the project when you are through.

23. The project is now working as expected except for the client-side JavaScript to tie the Washington and International radio buttons to the values in the Region column. Right-click the WebGrid and select Edit Layout. Set the following properties to tell the grid you will be responding to events to populate some of the controls on the row edit template, and returning user input from the row edit template back to the grid cells:

ClientSideEvents.TemplateUpdateCellsHandler = TemplateUpdateCells **ClientSideEvents.TemplateUpdateControlsHandler** = TemplateUpdateControls

The **TemplateUpdateControlsHandler** fires for each control on the template that has a grid column associated with it. But there is something missing. How are the radio buttons associated with the Region grid column? Currently, they are not. There is no property of the radio button to specify the related column key of the grid.

24. To associate a control on the row edit template with a specific column on the grid, a **columnKey** property must be added to the control. Select the WebForm1.aspx designer and click the HTML tab to display the underlying HTML. Now you must find the specifications for the controls on the row edit template by looking for the **RowEditTemplate** tag.
25. After locating this tag, take note of the **INPUT** tags for the ID, FirstName, and LastName text boxes. Also notice each of these controls has a **columnKey** property pointing to the column key value of the underlying WebGrid column. Continue down the text a couple of lines and you should see the **INPUT** tags for radio buttons. In both of these controls lines add the following property:

columnKey = Region

When you are complete, the HTML for the radio buttons should look something like (the following has been reformatted):

```
<DIV style="WIDTH: 240px; HEIGHT: 25px" ms_positioning="FlowLayout">
```

```
Region
```

```
<INPUT id="rbWashington" type="radio" value="rbWashington"
name="RadioGroup" columnKey="Region">
```

Washington

```
<INPUT id="rbInternational" type="radio" value="rbInternational"
name="RadioGroup" columnKey="Region">
```

International

```
</DIV>
```

26. With the columnKey properties specified, you are now ready to add some JavaScript to populate the radio buttons when the row template displays. Add the following JavaScript just above the </form> tag:

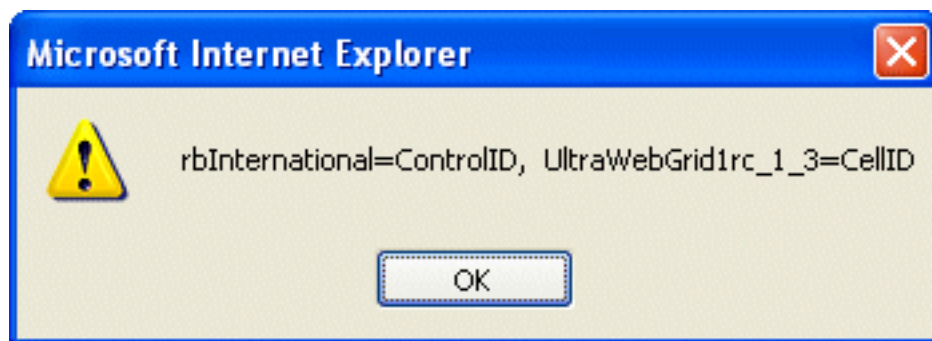
```
<script language="javascript">

function TemplateUpdateControls(gn,ctrlId,cellId,value)
{
    alert(ctrlId + "=ControlID, " + cellId + "=CellID")
}

function TemplateUpdateCells(gn,ctrlId,cellId)
{
    alert(ctrlId + "=ControlID, " + cellId + "=CellID")
}

</script>
```

27. This script displays a message box each time the event fires displaying the control ID and WebGrid Cell ID properties. Run the project, click on a row selector then click it again to display the row template. You will see five dialogs that look something like:



The controls are processed from bottom to top and as you can see from the above illustration the control ID for the International radio button is rbInternational Stop the project when you are done.

28. Replace the code in the TemplateUpdateControls event with the following to populate radio buttons based on values from the underlying row cell:

```
if(ctrlId=="rbWashington")
{
    if (value=="WA")
    {
        igtbl_getElementById(ctrlId).checked=true;
        return true;
    }
    else
    {
        igtbl_getElementById(ctrlId).checked=false;
    }
}
```

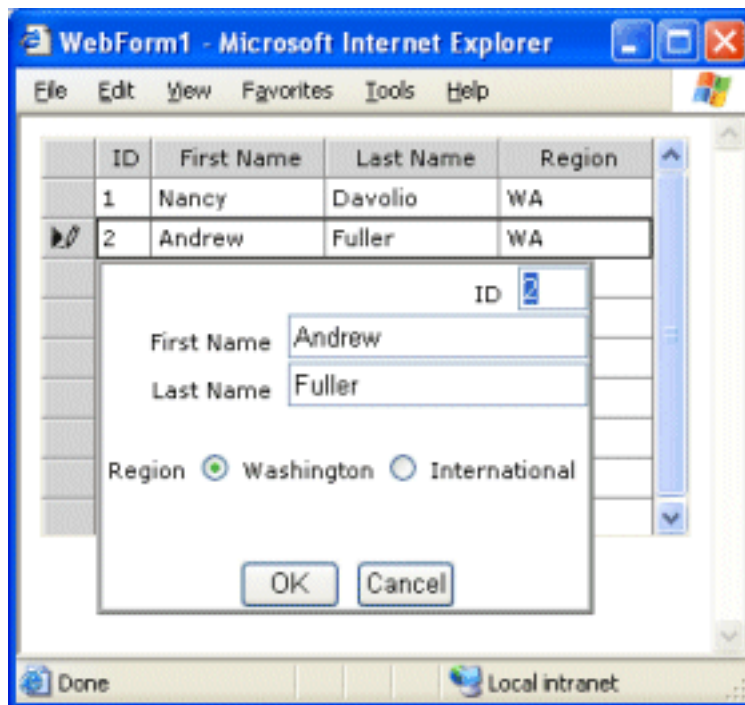


```

        return true;
    }
}
else if(ctrlId=="rbInternational")
{
    if (value=="IN")
    {
        igtbl_getElementById(ctrlId).checked=true;
        return true;
    }
    else
    {
        igtbl_getElementById(ctrlId).checked=false;
        return true;
    }
}
}

```

29. Run the project and click on one of the rows with a region value of WA, then click on the row selector again to display the row template. You should see something like:



Observe the region radio button for Washington is clicked. Stop the project when you are done.

30. Replace the code in the TemplateUpdateCells control to place the region cell values back into the WebGrid cells:

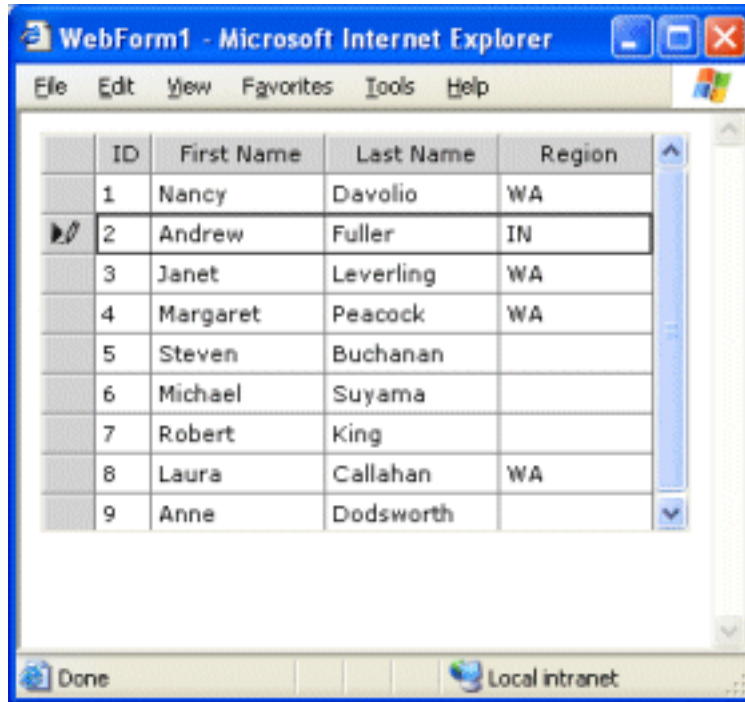
```

if(ctrlId=="rbWashington")
{
    if (igtbl_getElementById(ctrlId).checked==true)
    {
        igtbl_getCellById(cellId).setValue("WA");
        return true;
    }
}
else if(ctrlId=="rbInternational")
{
    if (igtbl_getElementById(ctrlId).checked==true)
    {
        igtbl_getCellById(cellId).setValue("IN");
    }
}

```

```
        return true;
    }
}
```

31. Run the project. Click on one of the rows then click it again to display the row template. Change the region setting and click OK. Notice the value of the Region column in the grid has change to reflect your new region setting:



Stop the project when you are finished.

Review

This tutorial shows how to use jScript to interface between WebGrid cells and a row template to provide a rich user input experience.

Server Events

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

Server Event Handlers have a profound effect on the WebGrid. Unlike the WinForms environment where adding an event handler has little to no performance implications. Adding an event handler for the WebGrid in the WebForms environment controls the amount of JavaScript rendered to the client and most significantly controls when automatic post backs will occur.

For example, setting the DisplayLayout.**AllowUpdateDefault** property to true and adding an event handler for the **UpdateCell** event will cause the WebGrid to perform a post back to the server each time a grid cell is changed.

Managing your server events will have a profound effect on application performance. Consider using the batch events instead of the individual cell events to reduce the number of server post backs.

Questions

- I am responding to the **UpdateCell** event on the server, but the grid performs a server post back each time a cell changes and the application performance is suffering. How do I fix this?

Solution

Use the batch version of the **Update**, **RowDelete** and **RowAdd** events and handle all of the user value changes on a single post back.

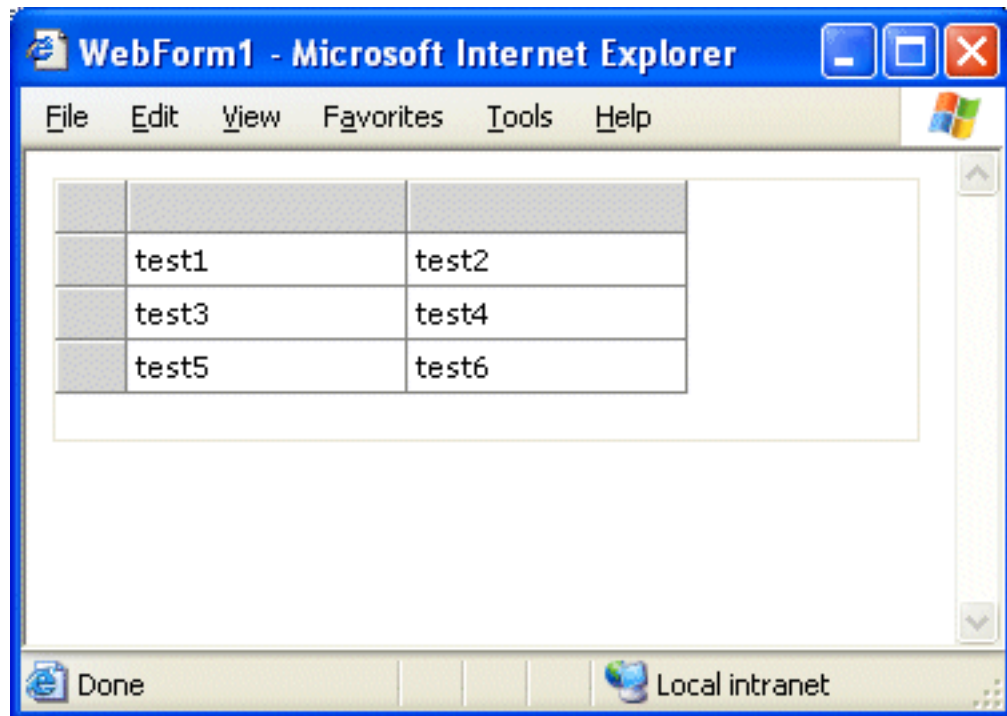
Sample Project

This sample project shows how adding an event handler to the grid will cause the grid to perform a server post back.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer, position and size as desired.
3. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.
4. Click the Infragistics.WebUI.Shared reference and set the following properties:
CopyLocal = True
5. Right-Click the WebGrid and select Edit Columns. Add two columns.
6. Click on Column 0 and set the Key property to Column 0.
7. Click on Column 1 and set the Key property to Column 1.
8. Click OK.

9. Right-click the WebGrid and select Edit Rows. Enter some text into three rows and click OK.
10. Run the project you should see something like:



Click around on the grid and notice that nothing you can do to the grid will cause a post back to the server. Then stop the project.

11. In the WebForm1.aspx designer, right-click the WebGrid and select Edit Layout. In the Edit DisplayLayout dialog, set the following property:

AllowUpdateDefault = Yes

12. Click Close and run the project. Again, click around on the grid and notice that nothing you can do to the grid will cause a post back, even though you can change the content of the cells. Stop the project when you are done.
13. From the Toolbox, Web Forms tab, add a text box below the grid. Click the Textbox and set the following properties:

Height = 100px

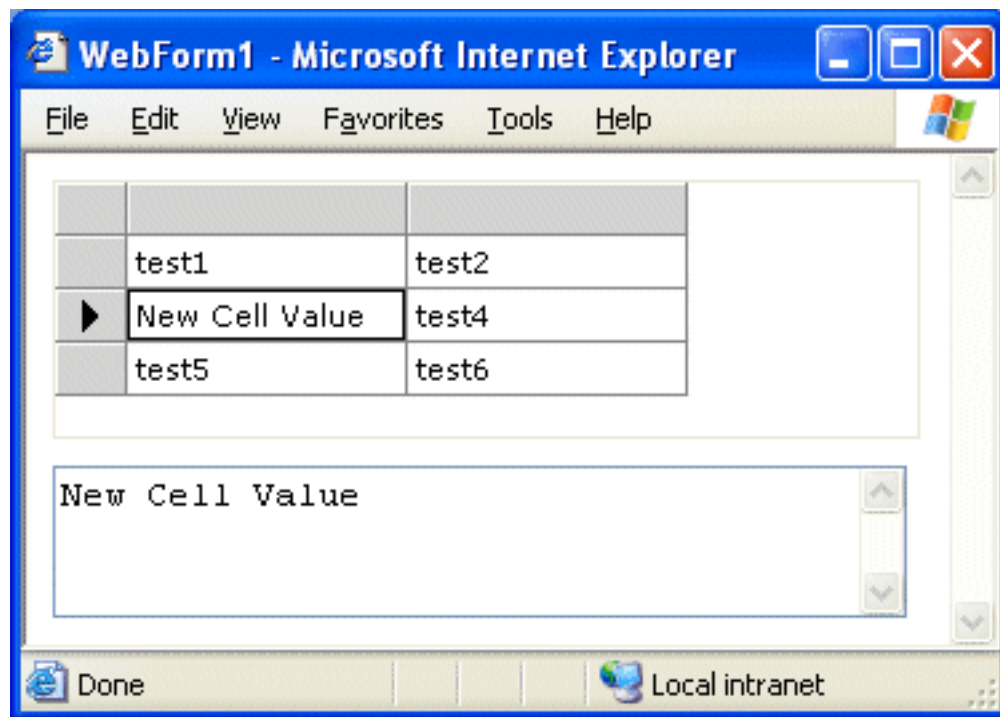
TextMode = MultiLine

14. Add the following code to the WebGrid UpdateCell event handler:

In Visual Basic:

```
' put the update cell value into the text box  
TextBox1.Text = e.Cell.Text
```

15. Run the project. Double-click on a cell and enter some text. Tab out of the cell or click on another cell. Notice the grid performed a post back to the server and the value you entered in the cell now displays in the text box.
As you can see, the simple act of adding an event handler to the project has caused the grid to perform a post back to the server every time a cell value changes.



Stop the project when you are done.

16. The following illustrates a **DbClick** event that's a little more interesting. When the **DbClick** event is implemented, the developer is presented with references to the Cell, Column and Row, and the ability to cancel the grid's built-in methods.

In this example, the cell value, column name, and row index will display whenever the **DbClick** event fires if there is a valid reference to the object.

Add the following code to the WebGrid DbClick event:

In Visual Basic:

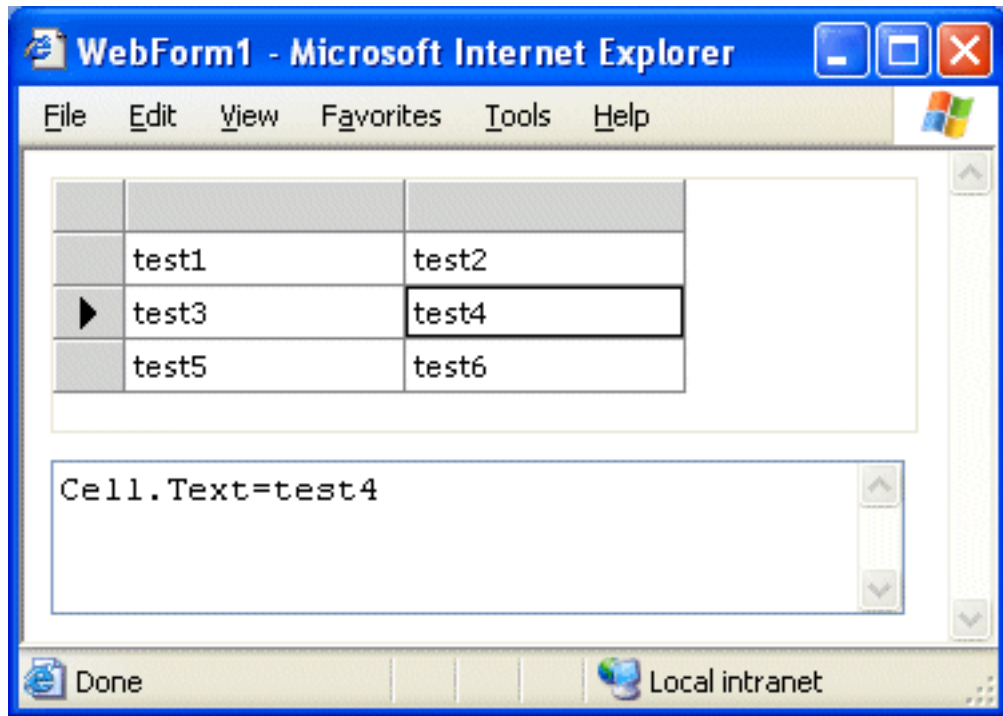
```
' clear text box text
TextBox1.Text = ""

' process cell
If Not e.Cell Is Nothing Then
    TextBox1.Text += "Cell.Text=" & e.Cell.Text & vbCrLf
End If

' process column
If Not e.Column Is Nothing Then
    TextBox1.Text += "Column.Key=" & e.Column.Key & vbCrLf
End If

' process row
If Not e.Row Is Nothing Then
    TextBox1.Text += "Row.Index=" & e.Row.Index.ToString & vbCrLf
End If
```

17. Run the project. Click around on the different visible elements and observe the values displayed in the text box:



Stop the project to end this tutorial.

Review

This sample project introduces the WebGrid server post back behavior experienced when a user input event handler is added to the project.

WebGrid On User Control

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

In some applications, there may be a situation where the WebGrid needs to be configured the same and populated the same on many different web forms.

For these applications, the WebGrid can be placed on a User Control along with all of the necessary initialization and configuration code, and the User Control placed on the various web forms.

Questions

- I have an application where I need the same WebGrid placed on many different forms. Can I place the WebGrid on a User Control?

Solution

Yes, place the WebGrid on a User Control just like you would on a Web Form. just realize there some intricate steps required to place User Controls on Web Forms.

Sample Project

This project places a WebGrid on a User Control and this user control is placed on multiple Web Forms. This methodology could be used to provide a list of items in a shopping cart as the user makes shopping cart item selections.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From the main menu select Project, Add Web User Control In the Name field type ShoppingCartControl.ascx and press Open.
3. From the Toolbox, HTML, drag and drop a Grid Layout Panel on the user control designer form. Size it to about 126px wide and 300px high.
4. From the Toolbox, HTML, drag and drop a Label on the panel. Move it to the top left corner and stretch it to the panel width. Click the label, then click again and change the text to "Shopping Cart."
5. In Properties, click style and open the style builder. Set the following properties:
 - Font.Family** = Verdana
 - Font.Color** = InactiveCaptionText
 - Font.Size.Absolute** = X-Small
 - Font.Bold.Absolute** = Bold
 - Background.Color** = ActiveCaption
 - Text.Alignment.Horizontal** = Centered
6. From the Toolbox, drag and drop a WebGrid onto the panel. Move to just under the label and stretch it to fill the remainder of the panel.
7. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.

Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.

- Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

- To supply the WebGrid with data, create a new Dataset:

- On Solution Explorer; Double-Click on WebForm1.aspx to open the web form designer.
- From the main menu; select Project, Add New Item, DataSet.
- Change the name to ShoppingCartDataset.xsd and click Open.
- Right-click the designer surface and select Add, New Element. Drag this new Element to the upper left corner of the designer.

- Change the element1 name to "Products".

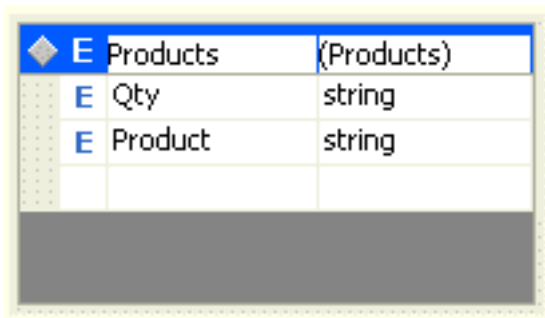
- Click the empty cell under the E in the top row. In the drop-down, select Element. Set the following properties:

name = Qty
type = string

- Click the empty cell under the E in the next row. In the drop-down select Element. Set the following properties:

name = Product
type = string

When completed the designer should look something like:



E Products (Products)	E Qty string	E Product string

- Select ShoppingCartControl.ascx. From Toolbox, Data, drag a DataSet onto the ShoppingCartControl.ascx designer.
- On the Add Dataset dialog, select Typed dataset and the .ShoppingCartDataset from the drop-down. This will create a typed dataset from an existing schema.
- Click on the WebGrid control and set the following properties:

DataSource = ShoppingCart1

DataMember = Products

DisplayLayout.RowSelectorDefault = No

DisplayLayout.ReadOnly = LevelZero

DisplayLayout.HeaderStyleDefault.BackColor = InactiveCaptionText

DisplayLayout.HeaderStyleDefault.ForeColor = ActiveCaption

DisplayLayout.RowStyleDefault.BorderColor = ActiveCaption

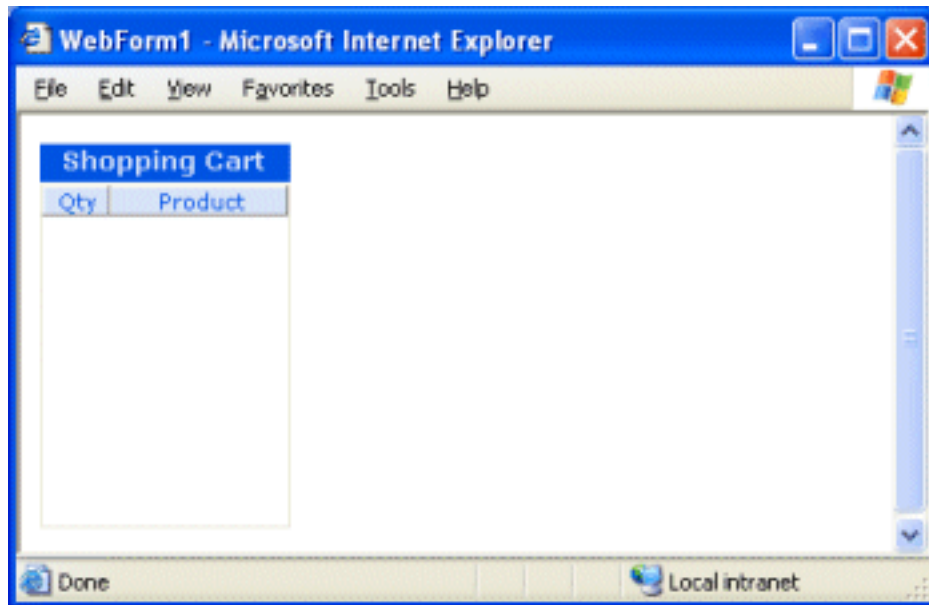
- Click the Columns (collection) property and open the Columns Collection Editor. Set the following properties:

Column 0 [Qty]:

Width = 30
CellMultiline = Yes

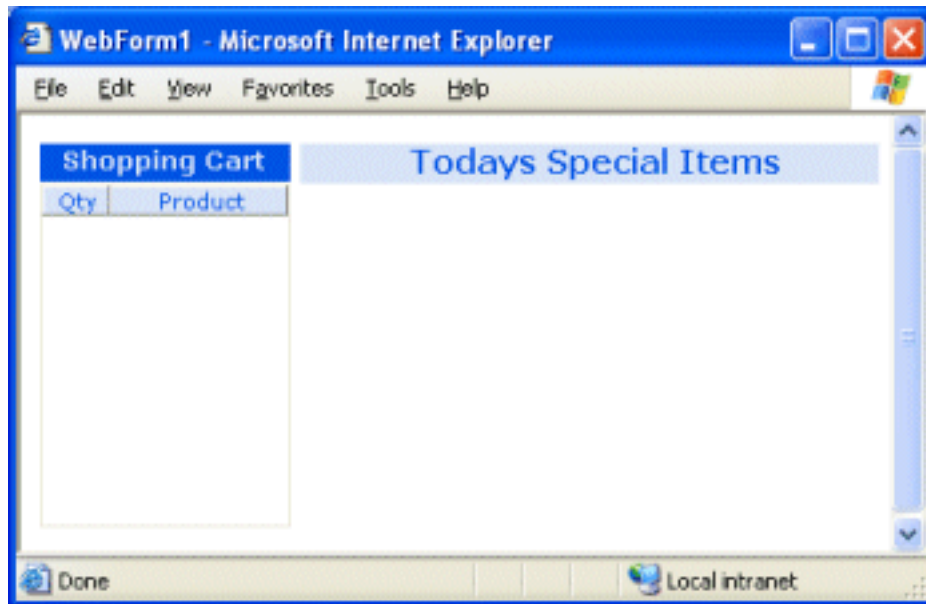
Column 1 [Product]:
Width = 90
CellMultiline = Yes

14. Add the ShoppingCart user control to the WebForm1 designer:
 - Select the WebForm1.aspx designer.
 - From the Solution Explorer; drag the ShoppingCartControl.ascx node to the designer and drop. The UserControl ShoppingCartControl1 should appear on the designer.
15. Run the project and you should see something like:



Stop the project.

16. Select the WebForm1.aspx designer. Add a header label: From Toolbox, HTML, add a label control. Click the label, and click it again and change the text to "Today's Special Items".
17. Click the Style property and open the Style Builder. Set the following properties:
 - Font.Family** = Verdana
 - Font.Color** = ActiveCaption
 - Font.Size.Absolute** = X-Small
 - Font.Bold.Absolute** = Bold
 - Background.Color** = InactiveCaptionText
 - Text.Alignment.Horizontal** = Centered
 - Text.Alignment.Vertical** = Centered
 - Position.Top** = 15px
 - Position.Left** = 140px
 - Position.Height** = 20px
 - Position.Width** = 365px
18. Run the project and you should see something like:



Stop the project.

19. Select the WebForm1.aspx designer. Add a three buttons used to add items to the shopping cart: From Toolbox, Web Forms, add three buttons and set the properties of each button as follows:

Button1:

ID = btnRedSocks
BackColor = InactiveCaptionText
Font.Bold = True
ForeColor = ActiveCaption
Text = "6 pair Red Socks"

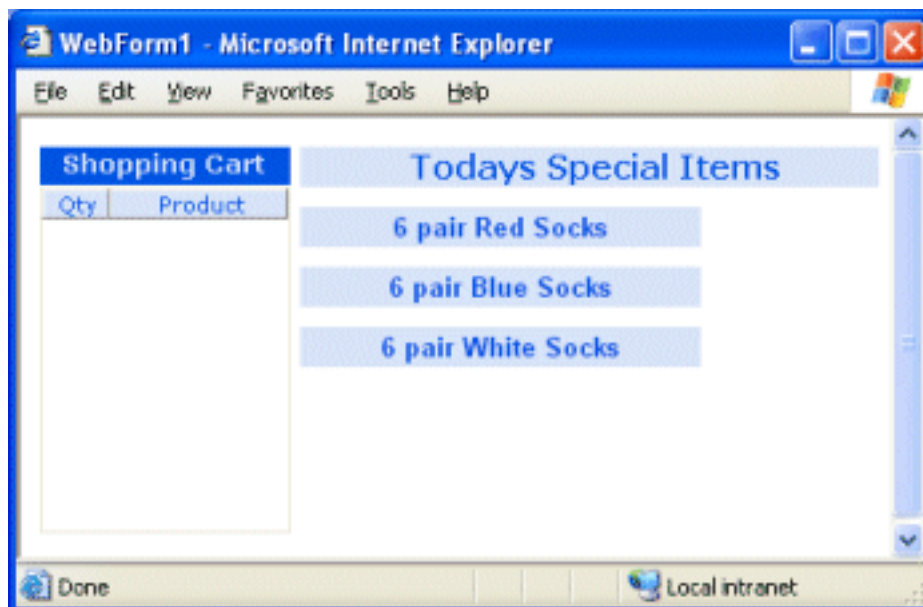
Button2:

ID = btnBlueSocks
BackColor = InactiveCaptionText
Font.Bold = True
ForeColor = ActiveCaption
Text = "6 pair Blue Socks"

Button3:

ID = btnWhite Socks
BackColor = InactiveCaptionText
Font.Bold = True
ForeColor = ActiveCaption
Text = "6 pair White Socks"

20. Run the project and you should see something like:



21. When the user control was added to the web form designer, there was no reference added by Visual Studio to provide access to the properties, methods and events of the control. Add the following code at the top the WebForm1.aspx.vb so you can have access to the code behind ShoppingCartControl1:

In Visual Basic:

```
Protected WithEvents ShoppingCartControl1 As ShoppingCartControl
```

22. Select the ShoppingCartControl.ascx.vb code behind and add the following method to add an item to the shopping cart:

In Visual Basic:

```
Public Sub AddItem(ByVal v_intQuantity As Integer , ByVal v_strProduct As String)
    ' declare a new row
    Dim wgrNew As New _
    Infragistics.WebUI.UltraWebGrid.UltraGridRow()

    ' delclare and populate a new quantity cell
    Dim wgcQty As New _
    Infragistics.WebUI.UltraWebGrid.UltraGridCell(CStr(v_intQuantity))

    ' declare and populate a new product cell
    Dim wgcProduct As New _
    Infragistics.WebUI.UltraWebGrid.UltraGridCell(v_strProduct)

    ' add the cells to the row
    wgrNew.Cells.Add(wgcQty)
    wgrNew.Cells.Add(wgcProduct)

    ' add the row to the grid
    UltraWebGrid1.Rows.Add(wgrNew)
End Sub
```

23. Select the WebForm1.aspx.vb code behind and add the following event handlers for the three button controls:

In Visual Basic:

```

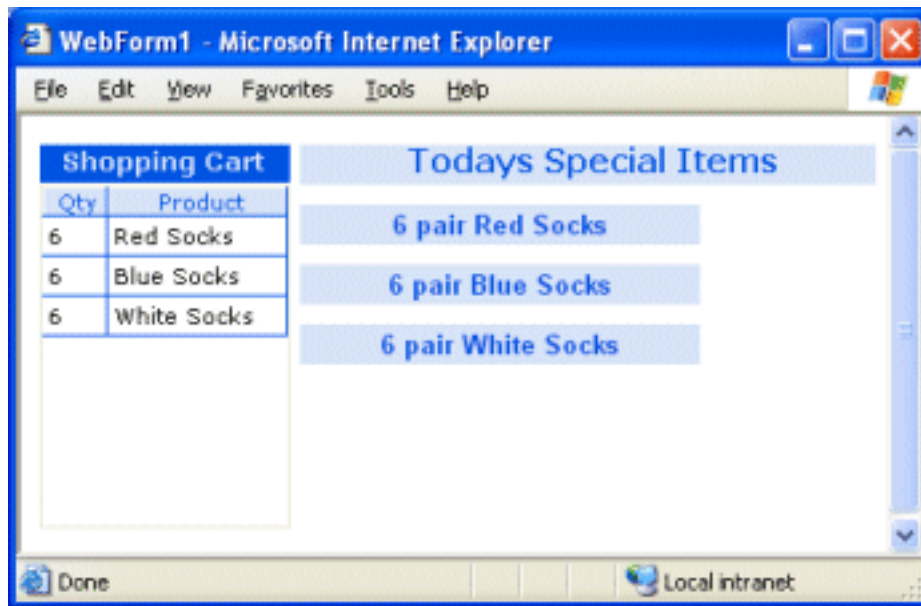
Private Sub btnRedSocks_Click(ByVal sender As Object _
, ByVal e As System.EventArgs) Handles btnRedSocks.Click
    ShoppingCartControl1.AddItem(6, "Red Socks")
End Sub

Private Sub btnBlueSocks_Click(ByVal sender As Object _
, ByVal e As System.EventArgs) Handles btnBlueSocks.Click
    ShoppingCartControl1.AddItem(6, "Blue Socks")
End Sub

Private Sub btnWhiteSocks_Click(ByVal sender As Object _
, ByVal e As System.EventArgs) Handles btnWhiteSocks.Click
    ShoppingCartControl1.AddItem(6, "White Socks")
End Sub

```

24. Run the project. Click on the three buttons and observe items being added to the shopping cart control:



25. To this point in the tutorial, there is no advantage over putting the WebGrid on the web form itself. Here is where the use of a user control starts to pay off.
- From the main menu; select Project, Add Web Form Change the name to NewItems.aspx and click open.
26. Select the WebForm1.aspx designer. With the mouse, select the label and the three buttons, right-click on one of the controls and select Copy.
27. Select the NewItems.aspx designer. With the mouse, right-click on the designer form and click Paste.
28. Drag all four controls to the right until their left position is 140px and their top position is 15px. You can see this position indicated in the status bar of Visual Studio as you move the controls. This will place the label and buttons in the same place as on web form 1.
29. Change the text of the label to "New Items".
30. Change the button ID properties and text to correspond to something like Pink Shirt, Grey Pants, and Yellow Socks.
31. From the Solution Explorer, drag the ShoppingCartControl.ascx node and drop it on the NewItems.aspx designer.
32. Add the following line of code to the NewItems.aspx.vb code behind so this form will have access to the members of the user control class:

In Visual Basic:

```
Protected WithEvents ShoppingCartControl1 As ShoppingCartControl
```

33. Add event handlers for the three buttons:

In Visual Basic:

```
Private Sub btnPinkShirt_Click(ByVal sender As Object _  
, ByVal e As System.EventArgs) Handles btnPinkShirt.Click  
    ShoppingCartControl1.AddItem(1, "Pink Shirt")  
End Sub
```

```
Private Sub btnGreyPants_Click(ByVal sender As Object _  
, ByVal e As System.EventArgs) Handles btnGreyPants.Click  
    ShoppingCartControl1.AddItem(1, "Grey Pants")  
End Sub
```

```
Private Sub btnYellowSocks_Click(ByVal sender As Object _  
, ByVal e As System.EventArgs) Handles btnYellowSocks.Click  
    ShoppingCartControl1.AddItem(1, "Yellow Socks")  
End Sub
```

34. Add a HyperLink control to web form 1 pointing to NewItems.aspx: Select the WebForm1.aspx designer. From the Toolbox, Web Forms tab, drag and drop a HyperLink control on the designer and position it below the existing buttons. Set the properties for this Hyperlink:

Text = "New Items"
NavigateUrl = NewItems.aspx

35. Add a HyperLink control to the new items form pointing to WebForm1.aspx: Select the NewItems.aspx designer. From the Toolbox, Web Forms tab, drag and drop a HyperLink control on the designer and position it below the existing buttons. Set the properties for this Hyperlink:

Text = Home
NavigateUrl = WebForm1.aspx

36. Run the project. Select some items on the home form, then click on the New Items link and notice the items selected from the home form are not present on the New Items form. Select some items on the New Items form and click the Home link, and notice the items selected originally on the home page are gone.

This is the expected behavior because you are not doing anything to save and coordinate the Dataset.

37. Modify the project to save the dataset in a session variable to persist the shopping cart contents. Select the ShoppingCartControl.ascx designer, click the web grid and set the following properties (some may already be set):

Columns(0).IsBound = False
Columns(1).IsBound = False
DataSource = *(remove this property setting)*
DataMember = *(remove this property setting)*
DisplayLayout.AutoGenerateColumns = False

38. Select the ShoppingCartControl.ascx.vb file editor. Add the following class scope declaration to hold the data set:

In Visual Basic:

```
Private c_dsShoppingCart As DataSet
```

39. Replace the **AddItem** method with the following:

In Visual Basic:

```
Public Sub AddItem(ByVal v_intQuantity As Integer _
, ByVal v_strProduct As String)
    ' declare a new row
    Dim drNew As ShoppingCartDataset.ProductsRow _
    = c_dsShoppingCart.Tables("Products").NewRow

    ' set cell values
    drNew.Qty = CStr(v_intQuantity)
    drNew.Product = CStr(v_strProduct)

    ' add the row to table
    c_dsShoppingCart.Tables("Products").Rows.Add(drNew)
End Sub
```

40. In the page's **Load** event add the following code:

In Visual Basic:

```
' retrieve or create shopping cart dataset
c_dsShoppingCart = Session("dsShoppingCart")
If c_dsShoppingCart Is Nothing Then
    c_dsShoppingCart = New ShoppingCartDataset()
End If
```

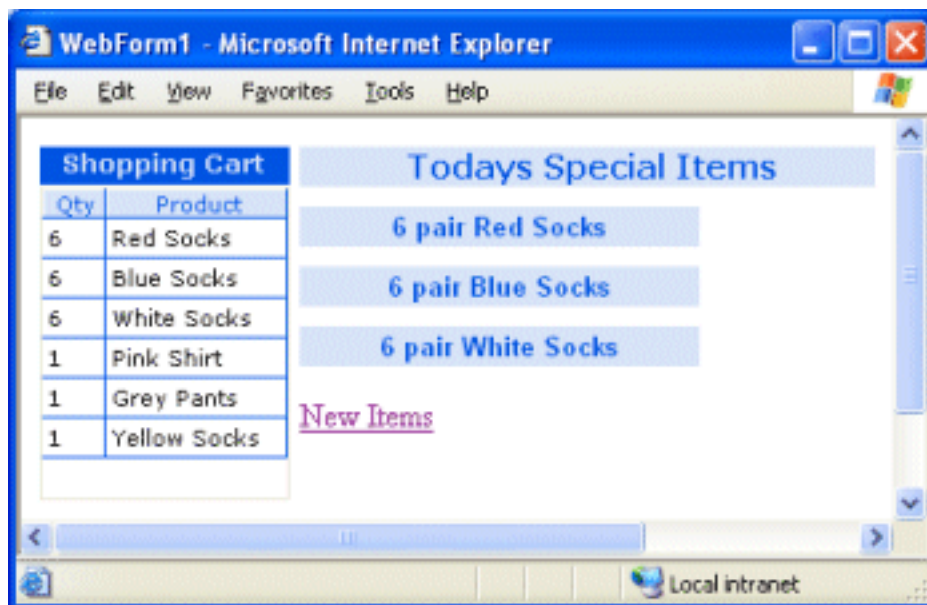
41. In the **PreRender** event add the following code:

In Visual Basic:

```
' set datasource and bind data
UltraWebGrid1.DataSource = c_dsShoppingCart.Tables("Products")
UltraWebGrid1.DataBind()

' save dataset as session variable
Session.Add("dsShoppingCart", c_dsShoppingCart)
```

42. Run the project and add some items from the main page and the new items page. Click the link buttons and jump between the main page and new items page and observe the grid retains items added on each page.



Stop the project.

Now, imagine a project where there are hundreds of different product pages. The shopping cart user control could be placed on each of these pages and encapsulate all of the shopping cart grid functionality. And, with the shopping cart dataset in a session variable, the information is readily available for check out.

Review

This tutorial shows how to place a WebGrid on a UserControl and make the properties, methods and events of the user control available to the code behind for the container form. Also illustrated is how to retain grid contents in a session variable.

The results of this tutorial can be expanded quickly into a complete shopping cart system with check out.

WebCombo On A Form

There are 5 major steps to this exercise:

- [Background](#)
- [Questions](#)
- [Solution](#)
- [Sample Project](#)
- [Review](#)

Background

The WebCombo primarily provides multi-column drop-down capabilities to the WebGrid. However, it can be used as a stand-alone control on a WebForm also. This tutorial shows how the WebCombo control can provide multi-column drop-down capabilities on a WebForm.

Questions

- I know the WebCombo provides multi-column drop-down capabilities for the WebGrid, but can I also use it on a WebForm?

Solution

Use the WebCombo on a WebForm to provide a multi-column drop-down user interface.

Sample Project

This project uses the Northwind Employees table as a source of list items for the WebCombo.

Perform the following steps to create this project:

1. Start a new Web Forms project.
2. From Toolbox, Data drag an OleDbDataAdapter to the WebForm1.aspx designer and complete the Data Adapter Configuration Wizard as follows:
 - On Choose Your Data Connection: select a known data connection or click New Connection and complete the Data Link Properties dialog to create a connection to the Access database. Typically the NWind.mdb file will be at a path something like `C:\Program Files\Infragistics\UltraWebGrid\v2.00.5000\Samples\Data\NWind.mdb`
 - Choose a Query Type: User SQL statements.
 - Generate the SQL Statements: Click Query Builder, Add Order Details and click Close, click (All Columns) and click OK.
 - Click Finish.
3. From the main Menu select Data, Generate Dataset and click OK From the Toolbox, drag an UltraWebGrid to the WebForm1.aspx designer. This will add the Register directive for the grid to the top of the aspx page which is needed when using the WebCombo designers.
4. Delete the UltraWebGrid.
5. From the Toolbox, drag a WebCombo to the WebForm1.aspx designer, position and size as desired.
6. In the Solution Explorer expand the References node. Look for a reference to Infragistics.WebUI.Shared. If the reference is not included, right-click the References node, select Add Reference, and add the reference from the .NET tab.

7. Click the Infragistics.WebUI.Shared reference and set the following properties:

CopyLocal = True

8. Click on the WebCombo and set the following properties:

DataSource = DataSet11

DataMember = Employees

DataValueColumn = EmployeeID

DisplayValueColumn = FirstName

9. Click the Columns (Collection) property and open the Column Collection Editor. Use the Up/Down arrow buttons to change the column positions to:

FirstName

LastName

BirthDate

EmployeeID

10. Select BirthDate and set the **Format** property to Date or MM/dd/yyyy.

11. Select EmployeeID and set the **Hidden** property to True.

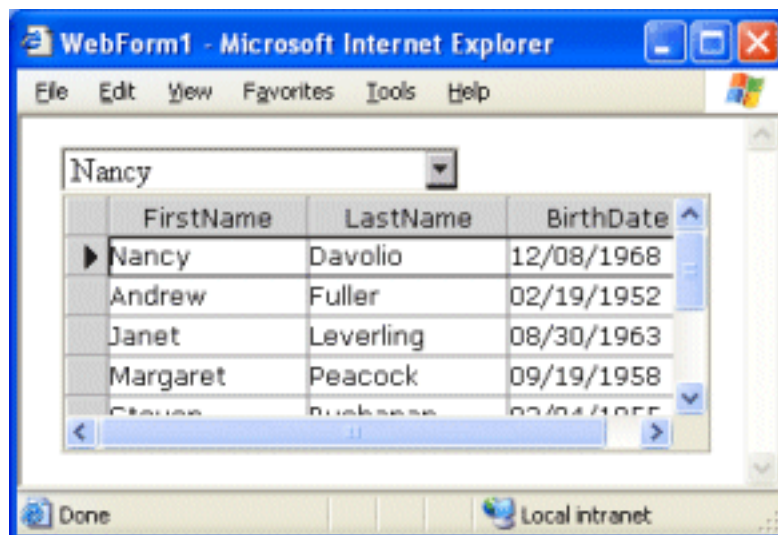
12. Click OK.

13. Add the following code to the page's **Load** event to fill the dataset and bind it to the WebGrid:

In Visual Basic:

```
If Me.IsPostBack = False Then
    Me.OleDbDataAdapter1.Fill(DataSet11)
    Me.WebCombo1.DataBind()
End If
```

14. Run the project and depending on which version of the Northwind database you connect to you should see something like:



Review

This project shows how to place and configure the WebCombo control on a web form.

Infragistics.WebUI.UltraWebGrid Namespace

[Inheritance Hierarchy](#)

Classes

Class	Description
ActivationObject	The ActivationObject object specifies the properties that will be applied to an object in the grid to indicate that it is the active object. The active object is the object that has input focus.
AddNewBox	The AddNewBox class handles properties and methods directly related to the AddNewBox button, which can be used to add a new row to the grid. The AddNewBox contains one AddNewButton for each Band of data in the grid. A new row can only be added to a band that already contains the ActiveRow or ActiveCell. The new row will be added at the end of the current data island. If the grid only contains 1 band, then the AddNewBox will contain one button and clicking it will cause a new row to be added at the bottom of the grid.
BandEventArgs	Event arguments that are passed to event handlers that involve band initialization.
BandsCollection	The collection of UltraGridBand objects.
BrowserChangedEventArgs	
CellEventArgs	Event arguments that are passed to event handlers that involve changes to cells.
CellItem	The CellItem object is used to implement cell edit templates in a templated column. The CellItem functions as the container into which the cell editing templates are instantiated.
CellsCollection	A collection of UltraGridCell objects that make up a row in teh grid.
ChangedCellPair	
ClickEventArgs	Event arguments that are passed to event handlers that involve mouse clicks.
ClientSideEvents	
ColumnControlIDEditor	Editor that chooses an IProvidesEmbeddableEditor implementation from those available on the current Web Form on which this Control exists.
ColumnControlIDEditor.ConsumerImpl	
ColumnDataType	Type converter for column's DataType
ColumnEventArgs	Event arguments that are passed to event handlers that involve changes to columns.
ColumnsCollection	The Columns collection manages the set of UltraGridColumn objects that correspond to a Band within the grid.
EditorControlIDEditorBase	General-purpose implementation of an Editor for choosing an IProvidesEmbeddableEditor control current residing on this Web Form for use as an Editor of any arbitrary object, component or control.

ExpandEffects	This object encapsulates the Internet Explorer Transitions functionality that UltraWebGrid's expandable objects expose.
FooterEventArgs	Event arguments that are passed to event handlers that involve footers.
FooterItem	The FooterItem object is used to implement footer templates in a templated column. The FooterItem functions as the container into which the column footer template is instantiated.
GridItemStyle	The GridItemStyle class handles properties and methods directly related to the appearance of an object inside the grid. GridItemStyles are applied to the cells, footers and column headers of the grid.
GroupByBox	The GroupByBox object represents the area at the top of the grid that is used for dragging or clicking column headers in order to group rows by the common values of that column.
GroupByRow	The GroupByRow object is a row that represents a grouped column. All rows in the data source that have common/equal values in the column that is grouped will be children of the GroupByRow object. The GroupByRow object contains a reference to the grouped column as well as a reference to the particular value that the child rows have in common.
HeaderItem	The HeaderItem object is used to implement header templates in a templated column. The HeaderItem functions as the container into which the column header template is instantiated.
ImageUrls	
InGridRenderer	Plug-in renderer implementation that allows extensions to render themselves in special docking regions within the UltraWebGrid.
InvalidCellsEnumerator	Enumerator for invalid cells in a grid.
LayoutEventArgs	Event arguments that are passed to event handlers that involve layout initialization.
PageEventArgs	Event arguments that are passed to event handlers that involve paging.
Pager	<p>The Pager class is responsible for controlling the paging behavior and appearance within the UltraWebGrid. There are three types of paging supported: 1. Built-in paging, Standard paging, and Custom paging. Built-in paging is completely automatic and does not require any coding to implement. Built-in paging is automatically enabled when the DisplayLayout.</p> <p>EnableInternalRowsManagement property is set to true. With this style of paging, however, there is a large ViewState requirement, as all pages of the query must be available to the grid without the need for a call to DataBind(). Standard paging simply requires that the application responds to the PageIndexChanged event to set the new page number and call DataBind(). UltraWebGrid indexes into the DataSource and finds the correct starting point and loads the grid with the PageSize number of rows. With Custom paging, it is the application's responsibility to provide a DataSource that has the exact set of rows to be displayed. UltraWebGrid will start at the beginning</p>

	of the DataSource and load the PageSize number of rows. In this scenario, the DataSource query has to be adjusted to return the correct set of rows, based upon the PageIndex or some other criteria.
RendererConsumerBase	Base class implementing IPlugInConsumer that supports pluggable external renderers for UltraWebGrid.
RowEventArgs	Event arguments that are passed to event handlers that involve changes to rows.
RowsCollection	The Rows Collection manages a set of UltraGridRow objects which are the child rows of an individual Row.
RowsEditor	Summary description for ColumnsEditor.
SelectedCellsCollection	This collection manages the UltraGridColumn objects that have been selected with the mouse on the client.
SelectedCellsEventArgs	Event arguments that are passed to event handlers that involve the selection of cells.
SelectedColsCollection	This collection manages the UltraGridRow objects that have been selected with the mouse on the client.
SelectedColumnsEventArgs	Event arguments that are passed to event handlers that involve the selection of columns.
SelectedRowsCollection	This collection manages the UltraGridColumn objects that have been selected with the mouse on the client.
SelectedRowsEventArgs	Event arguments that are passed to event handlers that involve the selection of rows.
SortColumnEventArgs	Event arguments that are passed to event handlers that involve column sorting.
SortedColsCollection	The collection of UltraGridColumn objects that are being used to sort the data in the band.
SpecialBoxBase	Base class for all box classes. It contains all common properties/methods
Strings	This class provides a common location for string that are used in the grid.
TemplatedColumn	The TemplatedColumn class provides access to the template-specific features of the UltraGridColumn object. You can use this class to manage the header, footer and cell templates attached to any given column.
UltraGridBand	The UltraGridBand object represents a single level of a hierarchical record set. The columns that make up a band are typically drawn from a single recordsource (table) in the data source.
UltraGridColumn	This class represents a cell in the grid (a specific column in a specific row).
UltraGridColumn	The Column object contains the properties and methods that control the look and behavior of a vertical column of cells within UltraWebGrid.
UltraGridEventArgs	The base class for UltraWebGrid events. It includes a Cancel property which allows the processing of the event to be stopped.
UltraGridLayout	
UltraGridRow	The UltraGridRow displays a row of data in the grid. A row typically represents the data from a single record in the attached datasource.
UltraGridRowsEnumerator	Enumerator for rows in a band.

UltraWebGrid	UltraWebGrid is the top-level object for the control. It provides access to the major functional areas of the control. The DisplayLayout property of UltraWebGrid contains many of the objects used to define behavior and appearance characteristics for the control. The Rows collection property allows access to the top level set of rows of the grid. If there are multiple levels, or Bands, for the grid, each of the top-level rows will allow access to any child rows that may be contained. For basic single level grids, UltraWebGrid exposes the Columns collection and the Rows collection for simple programmable access.
UpdateEventArgs	Event arguments that are passed to event handlers that involve cell updates.
ValidatorItem	
ValidatorItemsCollection	Summary description for ValidatorItems.
ValueList	The ValueList object maintains a list of values that the user can choose from when entering data. The ValueList is the common mechanism used to populate simple dropdown lists.
ValueListItem	The ValueListItem object represents a single row within the dropdown ValueList collection. The object manages the information for displaying and updating the item within the browser.
ValueListItemsCollection	The collection of ValueListItem objects that make up a ValueList object. This collection manages the adding, inserting and removal of ValueListItem elements from the list.
XmlClientWrite	Summary description for XmlClientWrite.

Interfaces

Interface	Description
ICanEditNavBar	
IControlEditorConsumer	Interface that supports reuse of an Editor Control infrastructure for arbitrary components on a Web Form.
IPlugInConsumer	Interface permitting registration of external plug-ins.
IPlugInFactory	Interface permitting implementations of plug-in interfaces to be created.
IPlugInRender	An interface that defines the services an external plug-in must provide to UltraWebGrid to inject customized HTML into the rendering.
IResolveStyles	Style resolution interface.
IUltraGridExporter	Interface for exporting UltraWebGrid into alternative formats.

Enumerations

Enumeration	Description
Activation	Determines how the cell will behave when it is activated.
AddNewBoxView	Indicates how the AddNewBox is rendered.
AllowAddNew	Enumeration used for controlling whether rows can be added or not on the client.
AllowColumnMoving	Used for defining whether the grid columns can be moved or not.

AllowDelete	Enumeration used for controlling whether rows can be deleted or not on the client.
AllowEditing	Enumeration used for controlling whether cells can be edited on the client.
AllowGroupBy	Enumeration used for setting the AllowGroupBy property of the DisplayLayout object.
AllowSizing	Indicates types of column and row resizing.
AllowSorting	Enumeration used for controlling whether sorting can be performed.
AllowUpdate	Enumeration used for controlling whether cells can be updated on the client.
BorderCollapse	Used for defining whether the grid borders should collapse or not.
BoxLocation	Enumeration used for setting the location of the AddNewBox.
BrowserLevel	Enumeration of the types of browser output supported.
BrowserTarget	Enumeration of the types of browser output supported.
Case	Indicates the case of text in a column.
CellButtonDisplay	Indicates when a cell's button is displayed.
CellClickAction	Indicates how a cell should react to being clicked.
CellMultiline	Enumeration for the Column. CellMultiline property. Determines whether cell editing will support multi-line text areas for text input.
ColumnType	Indicates the different column styles.
Cursors	Enumeration of cursors that can be used in styles.
DataChanged	Used for defining what kind of changes were made with a row.
DataType	Enumeration used for setting.
Expandable	Enumeration used for controlling whether rows are expandable or not.
ExpandEffectType	Enumeration of the various types of transitioning supported by UltraWebGrid when showing expandable objects. The Duration property of the ExpandEffects object controls how long each of these effects takes to complete.
HeaderClickAction	Indicates how a column or group header should react to being clicked.
LoadOnDemand	Enumeration of the types of Load On Demand styles supported. These property settings control the methods used to load rows into the grid.
Nullable	Indicates the way null values are stored.
PagerAlignment	Enumeration used for setting the alignment of the paging labels.
PagerAppearance	Enumeration used for setting the location of the paging labels.
PagerStyleMode	Enumeration used for setting the style of paging labels within the paging area of the grid.
ReadOnly	Used for defining what level of additional content that will be inserted into final HTML code.
RenderLocation	Enumeration used for setting the location where a plug-in renderer injects HTML.
RowSelectors	Enumeration values for row selectors presence.

RowsEnumMode	
ScrollBar	Enumeration used for controlling scrollbar appearance.
ScrollBarView	Enumeration used for controlling scrollbar view.
SelectType	Indicates the type of selection that is allowed for an object.
ShowBandLabels	Indicates how band labels are displayed in the GroupBy box.
ShowMarginInfo	Enumeration used for setting the visibility of several different objects in the boundary areas of the grid. These include ColFootersVisible and ColHeadersVisible.
SortIndicator	Indicates the method used to sort a column.
StationaryMargins	Used for defining which part or parts of the grid remain stationary while scrolling.
StyleContext	Enumeration describing supported contexts of the style resolution interface.
SummaryInfo	Indicates column footer summary information.
TabDirection	Enumeration used for defining the direction in which the tab key moves through the cells in the grid.
TableLayout	Indicates the layout of the grid.
TextOverflow	Used to specify how cell text should be handled when text wrapping is not enabled.
UltraGridLines	Indicates grid lines.
ValueListDisplayStyle	Indicates which property of the ValueListItem will be displayed in each item of the dropdown list. This applies to the ValueListItemsCollection collection associated with each ValueList for various dropdown column styles.
ViewState	Used for managing granular ViewState functionality.
ViewType	Indicates the view type of the grid.

Structures

Structure	Description
ProcessRowParams	Parameter-passing structure for Row exporting activities.

Delegates

Delegate	Description
ActiveCellChangeEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.ActiveCellChange event, which is generated when the active cell is about to be changed.
ActiveRowChangeEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.ActiveRowChange event, which is generated when the active row is about to be changed.
AddRowBatchEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.AddRowBatch event, which is generated when one or more new row(s) is about to be added to the grid.

AddRowEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.AddRow event, which is generated when a new row is about to be added to the grid.
BrowserChangedEventHandler	
ClickCellButtonEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.ClickCellButton event, which is generated when a cell button is clicked.
ClickEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.Click event, which is generated when a row, a column or a cell is clicked.
CollapseRowEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.CollapseRow event, which is generated when a row is about to be collapsed.
ColumnMoveEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.Move event, which is generated when the column is being moved.
ColumnSizeChangeEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.ColumnSizeChange event, which is generated when the width of a column is being changed.
DbClickEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.DbClick event, which is generated when a row, a column or a cell is double clicked.
DeleteRowBatchEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.DeleteRow event, which is generated when a row is about to be deleted.
DeleteRowEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.DeleteRow event, which is generated when a row is about to be deleted.
DemandLoadEventHandler	Event handler for DemandLoad events.
ExpandRowEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.ExpandRow event, which is generated when a row is about to be expanded.
GroupColumnEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.GroupColumn event, which is generated when a column is about to be grouped.
InitializeBandEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.InitializeBand event, which is generated when a band needs to be configured during data binding.
InitializeFooterEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.InitializeFooter event, which is generated when footers need to be configured during data binding.
InitializeGroupByRowEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.InitializeGroupByRow event, which is generated when a group by row needs to be initialized during its creation.

InitializeLayoutEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.InitializeLayout event, which is generated when the grid's layout needs to be initialized during data binding.
InitializeRowEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.InitializeRow event, which is generated when a row needs to be initialized during data binding.
PageIndexChangedEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.PageIndexChanged event, which is generated when a new page of data is about to be displayed when paging is enabled.
RenderDelegate	Delegate representing the rendering behavior of an HTML server control.
RowSizeChangeEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.RowSizeChange event, which is generated when the height of a row is being changed.
SelectedCellsChangeEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.SelectedCellsChange event, which is generated when a cell or cells in the grid are about to be selected or unselected.
SelectedColumnsChangeEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.SelectedColumnsChange event, which is generated when a column or columns in the grid are about to be selected or unselected.
SelectedRowsChangeEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.UltraWebGrid.SelectedRowsChange event, which is generated when a row or rows in the grid are about to be selected or unselected.
SortColumnEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.UltraWebGrid.SortColumn event, which is generated when a column is about to be sorted.
TemplatedColumnRestoredEventHandler	
UnGroupColumnEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.UnGroupColumn event, which is generated when a column is about to be ungrouped.
UpdateCellBatchEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.UpdateCellBatch event, which is generated when a cell's value is about to be updated.
UpdateCellEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.UltraWebGrid.UpdateCell event, which is generated when a cell's value is updated.
UpdateGridEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.UpdateGrid event, which is generated when the grid is updated, meaning the value of a cell or cells changed, or rows were added or deleted.

See Also

[Infragistics.WebUI.UltraWebGrid.v3.1 Assembly](#)

ActivationObject Class

The ActivationObject object specifies the properties that will be applied to an object in the grid to indicate that it is the active object. The active object is the object that has input focus.

For a list of all members of this type, see [ActivationObject members](#).

Inheritance Hierarchy

[System.Object](#)

[Infragistics.WebUI.Shared.WebComponentBase](#)

Infragistics.WebUI.UltraWebGrid.ActivationObject

Syntax

```
[Visual Basic]
Public Class ActivationObject
    Inherits WebComponentBase
    Implements IStateManager, ISupportRollback
```

```
[C#]
public class ActivationObject : WebComponentBase, IStateManager, ISupportRollback
```

```
[JScript]
public class ActivationObject extends WebComponentBase implements
IStateManager, ISupportRollback
```

Remarks

The ActivationObject controls the look and behavior of the rectangle that appears around individual rows and cells as they are selected with the mouse or keyboard on the client. The Activation object provides visual feedback to the user telling him or her which cell is currently active within the grid. The ActivationObject is created automatically by UltraWebGrid.

Using the ActivationObject, the color, width and style of the Activation rectangle can be set. If the grid is meant to be read-only, the Activation rectangle can be turned off completely using the AllowActivation property.


See Also

[ActivationObject Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)









ActivationObject Class Members

[ActivationObject overview](#)








Public Constructors

 ActivationObject Constructor	The default constructor for the Activation object.
---	--


Public Properties

 AllowActivation	Enables the use of the ActivationObject object.
 BorderColor	The color that will be applied to the border of the active object.
 BorderDetails	Returns a reference to or sets a Infragistics.WebUI.Shared.BorderDetails object that allows to customize the borders of the activation object.
 BorderStyle	The style that will be applied to the border of the active object.
 BorderWidth	The width that will be applied to the border of the active object.
 HasBorderDetails	Returns a Boolean value that determines whether the BorderDetail property is currently set to a Infragistics.WebUI.Shared.BorderDetails object.
 HasChanges (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 IsTrackingViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	

Public Methods

 Clone	Creates an copy of the existing ActivationObject object.
 Commit (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 CopyFrom	Copies the property settings of the specified ActivationObject object to the current ActivationObject object.
 CreateBackup (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 Reset	Overridden. Resets all properties of the ActivationObject class to their default values.
 Rollback (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 ToString	Overridden. Returns a string representation of an ActivationObject object.

Protected Properties

 ViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
--	--

See Also

[ActivationObject Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

AddNewBox Class

The AddNewBox class handles properties and methods directly related to the AddNewBox button, which can be used to add a new row to the grid. The AddNewBox contains one AddNewButton for each Band of data in the grid. A new row can only be added to a band that already contains the ActiveRow or ActiveCell. The new row will be added at the end of the current data island. If the grid only contains 1 band, then the AddNewBox will contain one button and clicking it will cause a new row to be added at the bottom of the grid.

For a list of all members of this type, see [AddNewBox members](#).

Object Model

AddNewBox

ButtonStyle (GridItemStyle)

Style (GridItemStyle)

Inheritance Hierarchy

[System.Object](#)

[Infragistics.WebUI.Shared.WebComponentBase](#)

[Infragistics.WebUI.UltraWebGrid.SpecialBoxBase](#)

Infragistics.WebUI.UltraWebGrid.AddNewBox

Syntax

```
[Visual Basic]
Public Class AddNewBox
    Inherits SpecialBoxBase
    Implements IStateManager, ISupportRollback
```

```
[C#]
public class AddNewBox : SpecialBoxBase, IStateManager, ISupportRollback
```

```
[JScript]
public class AddNewBox extends SpecialBoxBase implements IStateManager, ISupportRollback
```

See Also

[AddNewBox Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)












AddNewBox Class Members

[AddNewBox overview](#)




Public Constructors

 AddNewBox Constructor	
--	--


Public Properties

 ButtonConnectorColor (Inherited from SpecialBoxBase)	Returns a reference to or sets a Color object that specifies the color of the button connector lines for an object.
 ButtonConnectorStyle (Inherited from SpecialBoxBase)	Returns a reference to or sets a BorderStyle object that specifies the style of the button connector lines for an object.
 ButtonStyle	Returns a reference to or sets a GridItemStyle object that determines how the AddNewBox box is rendered on the client.
 HasButtonStyle	Returns a Boolean value that determines whether the ButtonStyle property is currently set to a GridItemStyle object.
 HasStyle (Inherited from SpecialBoxBase)	Returns a Boolean value that determines whether the Style property is currently set to a GridItemStyle object.
 Hidden	Returns or sets a Boolean value that determines whether the AddNewBox box is hidden.
 IsTrackingViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 Location	Returns or sets a value that specifies where the AddNewBox box is rendered to the client.
 Prompt	Returns or sets a string of text that is displayed to the left of the AddNewBox box.
 Style (Inherited from SpecialBoxBase)	Returns a reference to or sets a GridItemStyle object that determines how a button object is rendered on the client.
 View	Returns or sets a value that determines the view style of the AddNewBox box.

Public Methods

 CopyFrom	Overridden. Duplicates the properties of the specified SpecialBoxBase into the instance of the AddNewBox class from which this method is invoked.
 Reset	Overridden. Resets all properties of the AddNewBox class to their default values.
 ToString	Overridden. Returns a string representation of an AddNewBox object.

Protected Properties

 ViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
--	--

See Also

[AddNewBox Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

© 2004 • Infragistics, Inc.

BandEventArgs Class

Event arguments that are passed to event handlers that involve band initialization.

For a list of all members of this type, see [BandEventArgs members](#).

Object Model

BandEventArgs

Band (UltraGridBand)

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)

Infragistics.WebUI.UltraWebGrid.BandEventArgs

Syntax

```
[Visual Basic]
Public Class BandEventArgs
    Inherits UltraGridEventArgs
```

```
[C#]
public class BandEventArgs : UltraGridEventArgs
```

```
[JScript]
public class BandEventArgs extends UltraGridEventArgs
```



See Also

[BandEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

BandEventArgs Class Members

[BandEventArgs overview](#)

Public Properties

 Band	A reference to the Band object that the event applies to.
 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.

See Also

[BandEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

BandsCollection Class

The collection of UltraGridBand objects.

For a list of all members of this type, see [BandsCollection members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

[System.Collections.CollectionBase](#)

[Infragistics.WebUI.UltraWebGrid.KeyedObjectCollectionBase](#)

Infragistics.WebUI.UltraWebGrid.BandsCollection

Syntax

```
[Visual Basic]
Public Class BandsCollection
    Inherits KeyedObjectCollectionBase
    Implements IList, ICollection, IEnumerable, IStateManager, ISupportRollback
```

```
[C#]
public class BandsCollection : KeyedObjectCollectionBase,
    IList, ICollection, IEnumerable, IStateManager, ISupportRollback
```

```
[JScript]
public class BandsCollection extends KeyedObjectCollectionBase implements
    IList, ICollection, IEnumerable, IStateManager, ISupportRollback
```

Remarks

Each UltraGridBand represents a level in the hierarchical organization of the grid, and corresponds to a table or similar recordsource in the datasource to which the grid is bound. For programmer convenience and design-time persistence, the Bands collections is available in two places within the grid: From the **Infragistics.WebUI.UltraWebGrid.Bands** property of the UltraWebGrid object and from the **Infragistics.WebUI.UltraWebGrid.DisplayLayout.Bands** property of the DisplayLayout object.


See Also

[BandsCollection Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)






BandsCollection Class Members

[BandsCollection overview](#)














Public Constructors

 BandsCollection Constructor	Public constructor for the Bands Collection. The Bands collection is created for the grid by default and does not need to be constructed.
--	---



Public Properties

 All (Inherited from KeyedObjectCollectionBase)	The collection as an array of objects
 Count (Inherited from System.Collections.CollectionBase)	
 IsReadOnly	Overridden. Returns a Boolean value indicating whether the collection is read-only.
 Item	Property indexer for the collection.
 Owner (Inherited from KeyedObjectCollectionBase)	Provides public access to the owning object of this collection
















Public Methods

 Add	Inserts an UltraGridBand object into the Bands collection.
 Clear (Inherited from System.Collections.CollectionBase)	Removes all UltraGridBand objects from the collection.
 Contains (Inherited from KeyedObjectCollectionBase)	Returns true if the collection contains this item
 CopyTo (Inherited from KeyedObjectCollectionBase)	Copies the items into the array
 Exists (Inherited from KeyedObjectCollectionBase)	Returns true if an object with this key is already in the collection. Note, if key is null or a zero length string this method returns false
 FromKey	Property indexer for the collection. Uses the Key string to return an UltraGridBand object.
 GetEnumerator	Returns an enumerator object for the collection.
 GetItem (Inherited from KeyedObjectCollectionBase)	Overloaded.
 IndexOf (Inherited from KeyedObjectCollectionBase)	Overloaded.
 Insert	Inserts an UltraGridBand object into the Bands collection at a specified index.
 Remove	Removes the specified UltraGridBand object from the collection.
 RemoveAt (Inherited from System.Collections.CollectionBase)	Removes the UltraGridBand object at the specified index from the collection.
 ValidateKeyDoesNotExist (Inherited from KeyedObjectCollectionBase)	Overloaded.

Protected Properties

 InnerList (Inherited from System.Collections.CollectionBase)	
 List (Inherited from System.Collections.CollectionBase)	

Protected Methods

 CreateArray (Inherited from KeyedObjectCollectionBase)	Virtual method used by the All 'get' method to create the array it returns.
 InternalAdd (Inherited from KeyedObjectCollectionBase)	Appends the object to the collection
 InternalClear (Inherited from KeyedObjectCollectionBase)	Clears the collection
 InternalInsert (Inherited from KeyedObjectCollectionBase)	Inserts an object into the collection
 InternalRemove (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 InternalRemoveAt (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 OnClear (Inherited from System.Collections.CollectionBase)	
 OnClearComplete (Inherited from System.Collections.CollectionBase)	
 OnInsert (Inherited from System.Collections.CollectionBase)	
 OnInsertComplete (Inherited from System.Collections.CollectionBase)	
 OnRemove (Inherited from System.Collections.CollectionBase)	
 OnRemoveComplete (Inherited from System.Collections.CollectionBase)	
 OnSet (Inherited from System.Collections.CollectionBase)	
 OnSetComplete (Inherited from System.Collections.CollectionBase)	
 OnValidate (Inherited from System.Collections.CollectionBase)	

See Also

[BandsCollection Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

BrowserChangedEventArgs Class

For a list of all members of this type, see [BrowserChangedEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.BrowserChangedEventArgs

Syntax

```
[Visual Basic]  
Private Class BrowserChangedEventArgs
```

```
[C#]  
private class BrowserChangedEventArgs
```

```
[JScript]  
private class BrowserChangedEventArgs
```


See Also

[BrowserChangedEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)



BrowserChangedEventArgs Class Members

[BrowserChangedEventArgs overview](#)

Public Constructors

 BrowserChangedEventArgs Constructor	Creates an instance of the BrowserChangedEventArgs that can be passed to listeners of the BrowserChanged event.
--	---

Public Properties

 NewLevel	New Browser Level.
 OldLevel	Old Browser Level.

See Also

[BrowserChangedEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

CellEventArgs Class

Event arguments that are passed to event handlers that involve changes to cells.

For a list of all members of this type, see [CellEventArgs members](#).

Object Model

CellEventArgs

└─ **Cell (UltraGridCell)**

Inheritance Hierarchy

[System.Object](#)

└─ [System.EventArgs](#)

└─ [Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)

└─ **Infragistics.WebUI.UltraWebGrid.CellEventArgs**

Syntax

```
[Visual Basic]
Public Class CellEventArgs
    Inherits UltraGridEventArgs
```

```
[C#]
public class CellEventArgs : UltraGridEventArgs
```

```
[JScript]
public class CellEventArgs extends UltraGridEventArgs
```




See Also

[CellEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

CellEventArgs Class Members

[CellEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 Cell	Returns the Cell object associated with the event.
 Data	Returns the Data object associated with the event.

See Also

[CellEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

CellItem Class

The CellItem object is used to implement cell edit templates in a templated column. The CellItem functions as the container into which the cell editing templates are instantiated.

For a list of all members of this type, see [CellItem members](#).

Object Model

CellItem

Cell (UltraGridCell)

Inheritance Hierarchy

[System.Object](#)

[System.Web.UI.Control](#)

Infragistics.WebUI.UltraWebGrid.CellItem

Syntax

```
[Visual Basic]
Public Class CellItem
    Inherits Control
    Implements
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, INamingContainer
```

```
[C#]
public class CellItem : Control,
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, INamingContainer
```

```
[JScript]
public class CellItem extends Control implements
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, INamingContainer
```

See Also

[CellItem Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

















CellItem Class Members

[CellItem overview](#)








Public Constructors

 CellItem Constructor	
---	--







Public Properties

 BindingContainer (Inherited from System.Web.UI.Control)	
 Cell	Returns the cell that will be stored in the data source. This property is read-only.
 ClientID (Inherited from System.Web.UI.Control)	Gets the server control identifier generated by ASP.NET.
 Controls (Inherited from System.Web.UI.Control)	Gets a System.Web.UI.ControlCollection object that represents the child controls for a specified server control in the UI hierarchy.
 EnableViewState (Inherited from System.Web.UI.Control)	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client.
 FormattedText	Returns the text of the cell as it will be displayed in the grid, with any masking or formatting applied. This property is read-only.
 ID (Inherited from System.Web.UI.Control)	Gets or sets the programmatic identifier assigned to the server control.
 NamingContainer (Inherited from System.Web.UI.Control)	Gets a reference to the server control's naming container, which creates a unique namespace for differentiating between server controls with the same System.Web.UI.Control.ID property value.
 Page (Inherited from System.Web.UI.Control)	Gets a reference to the System.Web.UI.Page instance that contains the server control.
 Parent (Inherited from System.Web.UI.Control)	Gets a reference to the server control's parent control in the page control hierarchy.
 Site (Inherited from System.Web.UI.Control)	Gets information about the Web site to which the server control belongs.
 TemplateSourceDirectory (Inherited from System.Web.UI.Control)	Gets the virtual directory of the System.Web.UI.Page or System.Web.UI.UserControl that contains the current server control.
 Text	Returns the plain text of the cell with any formatting or masking removed. If the user has entered data in the cell, this text will correspond to the characters entered by the user. This property is read-only.
 UniqueID (Inherited from System.Web.UI.Control)	Gets the unique, hierarchically-qualified identifier for the server control.
 Value	Returns the value of the cell that will be stored in the data source. This may be text or a pure value such as true/false. This property is read-only.
 Visible (Inherited from System.Web.UI.Control)	Gets or sets a value that indicates whether a server control is rendered as UI on the page.








Public Methods

 DataBind (Inherited from System.Web.UI.Control)	Binds a data source to the invoked server control and all its child controls.
 Dispose (Inherited from System.Web.UI.Control)	Enables a server control to perform final clean up before it is released from memory.
 FindControl (Inherited from System.Web.UI.Control)	Overloaded.
 HasControls (Inherited from System.Web.UI.Control)	Determines if the server control contains any child controls.
 RenderControl (Inherited from System.Web.UI.Control)	Outputs server control content to a provided System.Web.UI.HtmlTextWriter object and stores tracing information about the control if tracing is enabled.
 ResolveUrl (Inherited from System.Web.UI.Control)	Converts a URL into one that is usable on the requesting client.
 SetRenderMethodDelegate (Inherited from System.Web.UI.Control)	


Public Events

 DataBinding (Inherited from System.Web.UI.Control)	
 Disposed (Inherited from System.Web.UI.Control)	
 Init (Inherited from System.Web.UI.Control)	
 Load (Inherited from System.Web.UI.Control)	
 PreRender (Inherited from System.Web.UI.Control)	
 Unload (Inherited from System.Web.UI.Control)	

Protected Properties

 ChildControlsCreated (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control's child controls have been created.
 Context (Inherited from System.Web.UI.Control)	Gets the System.Web.HttpContext object associated with the server control for the current Web request.
 Events (Inherited from System.Web.UI.Control)	Gets a list of event handler delegates for the control. This property is read-only.
 HasChildViewState (Inherited from System.Web.UI.Control)	Gets a value indicating whether the current server control's child controls have any saved view-state settings.
 IsTrackingViewState (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control is saving changes to its view state.
 ViewState (Inherited from System.Web.UI.Control)	Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page.
 ViewStateIgnoresCase (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the System.Web.UI.StateBag object is case-insensitive.

Protected Methods

 AddedControl (Inherited from System.Web.UI.Control)	Called after a control is added to the System.Web.UI.Control.Controls collection of another control.
--	---

 AddParsedSubObject (Inherited from System.Web.UI.Control)	Notifies the server control that an element, either XML or HTML, was parsed, and adds the element to the server control's System.Web.UI.ControlCollection object.
 BuildProfileTree (Inherited from System.Web.UI.Control)	
 ClearChildViewState (Inherited from System.Web.UI.Control)	Deletes the view-state information for all the server control's child controls.
 CreateChildControls	Overridden.
 CreateControlCollection (Inherited from System.Web.UI.Control)	Creates a new System.Web.UI.ControlCollection object to hold the child controls (both literal and server) of the server control.
 EnsureChildControls (Inherited from System.Web.UI.Control)	Determines whether the server control contains child controls. If it does not, it creates child controls.
 IsLiteralContent (Inherited from System.Web.UI.Control)	Determines if the server control holds only literal content.
 LoadViewState (Inherited from System.Web.UI.Control)	Restores view-state information from a previous page request that was saved by the System.Web.UI.Control.SaveViewState method.
 MapPathSecure (Inherited from System.Web.UI.Control)	Retrieves the physical path that a virtual path, either absolute or relative, maps to.
 OnBubbleEvent (Inherited from System.Web.UI.Control)	Determines whether the event for the server control is passed up the page's UI server control hierarchy.
 OnDataBinding (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.DataBinding event.
 OnInit (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Init event.
 OnLoad (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Load event.
 OnPreRender (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.PreRender event.
 OnUnload (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Unload event. Server controls should perform any final cleanup, such as closing files, closing database connections, and discarding objects, during this stage of the server control lifecycle.
 RaiseBubbleEvent (Inherited from System.Web.UI.Control)	Assigns any sources of the event and its information to the control's parent.
 RemovedControl (Inherited from System.Web.UI.Control)	Called after a control is removed from the System.Web.UI.Control.Controls collection of another control.
 Render	Overridden.
 RenderChildren (Inherited from System.Web.UI.Control)	Outputs the content of a server control's children to a provided System.Web.UI.HtmlTextWriter object, which writes the content to be rendered on the client.
 SaveViewState (Inherited from System.Web.UI.Control)	Saves any server control view-state changes that have occurred since the time the page was posted back to the server.
 TrackViewState (Inherited from System.Web.UI.Control)	Causes tracking of view-state changes to the server control so they can be stored in the server control's System.Web.UI.StateBag object. This object is accessible through the System.Web.UI.Control.ViewState property.

See Also

[CellItem Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

© 2004 • Infragistics, Inc.

CellsCollection Class

A collection of UltraGridColumn objects that make up a row in the grid.
For a list of all members of this type, see [CellsCollection members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

[System.Collections.CollectionBase](#)

[Infragistics.WebUI.UltraWebGrid.KeyedObjectCollectionBase](#)

Infragistics.WebUI.UltraWebGrid.CellsCollection

Syntax

```
[Visual Basic]
Public Class CellsCollection
    Inherits KeyedObjectCollectionBase
    Implements IList, ICollection, IEnumerable, IStateManager
```

```
[C#]
public class CellsCollection : KeyedObjectCollectionBase,
    IList, ICollection, IEnumerable, IStateManager
```

```
[JScript]
public class CellsCollection extends KeyedObjectCollectionBase implements
    IList, ICollection, IEnumerable, IStateManager
```

Remarks

The Cells collection is created for each UltraGridRow in the grid. The UltraGridColumn objects in the Cells collection contain the contents of the cell as well as style information to control the appearance of the cell.


See Also

[CellsCollection Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)







CellsCollection Class Members

[CellsCollection overview](#)















Public Constructors

 CellsCollection Constructor	Overloaded.
--	-------------




Public Properties

 All (Inherited from KeyedObjectCollectionBase)	The collection as an array of objects
 Count (Inherited from System.Collections.CollectionBase)	
 IsReadOnly	Overridden. Returns a Boolean value indicating whether the collection is read-only.
 Item	Cell property indexer for the Cells collection. Uses numeric index.
 Owner (Inherited from KeyedObjectCollectionBase)	Provides public access to the owning object of this collection
 Row	Gets the Owner Row for the Cells collection.















Public Methods

 Add	Overloaded. Inserts an UltraGridColumn object into the collection.
 Clear (Inherited from System.Collections.CollectionBase)	Removes all UltraGridColumn objects from the collection.
 Contains (Inherited from KeyedObjectCollectionBase)	Returns true if the collection contains this item
 CopyTo (Inherited from KeyedObjectCollectionBase)	Copies the items into the array
 Exists (Inherited from KeyedObjectCollectionBase)	Returns true if an object with this key is already in the collection. Note, if key is null or a zero length string this method returns false
 FromKey	Cell property indexer for the Cells collection. Uses the column's Key string to return a cell.
 GetEnumerator	Returns an enumerator object for the Cells collection.
 GetItem (Inherited from KeyedObjectCollectionBase)	Overloaded.
 IndexOf (Inherited from KeyedObjectCollectionBase)	Overloaded.
 Insert	Inserts an UltraGridColumn object into the collection at the specified index.
 Remove	Removes the specified UltraGridColumn object from the collection.
 RemoveAt (Inherited from System.Collections.CollectionBase)	Removes the UltraGridColumn object at the specified index from the collection.
 SetAt	Replaces an UltraGridColumn object in the collection at the specified index.
 ValidateKeyDoesNotExist (Inherited from KeyedObjectCollectionBase)	Overloaded.

Protected Properties

 InnerList (Inherited from System.Collections.CollectionBase)	
 IsTrackingViewState	
 List (Inherited from System.Collections.CollectionBase)	

Protected Methods

 CreateArray (Inherited from KeyedObjectCollectionBase)	Virtual method used by the All 'get' method to create the array it returns.
 InternalAdd (Inherited from KeyedObjectCollectionBase)	Appends the object to the collection
 InternalClear (Inherited from KeyedObjectCollectionBase)	Clears the collection
 InternalInsert (Inherited from KeyedObjectCollectionBase)	Inserts an object into the collection
 InternalRemove (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 InternalRemoveAt (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 OnClear (Inherited from System.Collections.CollectionBase)	
 OnClearComplete (Inherited from System.Collections.CollectionBase)	
 OnInsert (Inherited from System.Collections.CollectionBase)	
 OnInsertComplete (Inherited from System.Collections.CollectionBase)	
 OnRemove (Inherited from System.Collections.CollectionBase)	
 OnRemoveComplete (Inherited from System.Collections.CollectionBase)	
 OnSet (Inherited from System.Collections.CollectionBase)	
 OnSetComplete (Inherited from System.Collections.CollectionBase)	
 OnValidate (Inherited from System.Collections.CollectionBase)	

See Also

[CellsCollection Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ChangedCellPair Class

For a list of all members of this type, see [ChangedCellPair members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)
Infragistics.WebUI.UltraWebGrid.ChangedCellPair

Syntax

```
[Visual Basic]  
Public Class ChangedCellPair
```

```
[C#]  
public class ChangedCellPair
```

```
[JScript]  
public class ChangedCellPair
```


See Also

[ChangedCellPair Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)



ChangedCellPair Class Members

[ChangedCellPair overview](#)

Public Constructors

 ChangedCellPair Constructor	Overloaded.
--	-------------

Public Properties

 Cell	
 ChangedValue	

See Also

[ChangedCellPair Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ClickEventArgs Class

Event arguments that are passed to event handlers that involve mouse clicks.

For a list of all members of this type, see [ClickEventArgs members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)
[System.EventArgs](#)
[Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)
Infragistics.WebUI.UltraWebGrid.ClickEventArgs

Syntax

```
[Visual Basic]  
Public Class ClickEventArgs  
    Inherits UltraGridEventArgs
```

```
[C#]  
public class ClickEventArgs : UltraGridEventArgs
```

```
[JScript]  
public class ClickEventArgs extends UltraGridEventArgs
```





See Also

[ClickEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ClickEventArgs Class Members

[ClickEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 Cell	Returns the Cell object associated with the event. If the click event does not occur over a cell, this property will be null.
 Column	Returns the Column object associated with the event. If the click event does not occur over a column or column header, this property will be null.
 Row	Returns the Row object associated with the event. If the click event does not occur over a row, this property will be null.

See Also

[ClickEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ColumnControlIDEditor Class

Editor that chooses an **IProvidesEmbeddableEditor** implementation from those available on the current Web Form on which this Control exists.

For a list of all members of this type, see [ColumnControlIDEditor members](#).

Inheritance Hierarchy

[System.Object](#)

[System.Drawing.Design.UITypeEditor](#)

[Infragistics.WebUI.UltraWebGrid.EditorControlIDEditorBase](#)

Infragistics.WebUI.UltraWebGrid.ColumnControlIDEditor

Syntax

```
[Visual Basic]
Public Class ColumnControlIDEditor
    Inherits EditorControlIDEditorBase
```

```
[C#]
public class ColumnControlIDEditor : EditorControlIDEditorBase
```

```
[JScript]
public class ColumnControlIDEditor extends EditorControlIDEditorBase
```


See Also

[ColumnControlIDEditor Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)





ColumnControlIDEditor Class Members

[ColumnControlIDEditor overview](#)

Public Constructors

 ColumnControlIDEditor Constructor	Called by the design-time environment to instantiate an Editor, not intended for use from application logic.
--	--

Public Methods

 EditValue (Inherited from EditorControlIDEditorBase)	Overloaded.
 GetEditStyle (Inherited from EditorControlIDEditorBase)	Overloaded.
 GetPaintValueSupported (Inherited from EditorControlIDEditorBase)	Overloaded.
 PaintValue (Inherited from System.Drawing.Design.UITypeEditor)	Overloaded.

See Also

[ColumnControlIDEditor Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ColumnControlIDEditor.ConsumerImpl Class

For a list of all members of this type, see [ColumnControlIDEditor.ConsumerImpl members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.ColumnControlIDEditor.ConsumerImpl

Syntax

```
[Visual Basic]
Public Class ColumnControlIDEditor.ConsumerImpl
    Implements IControlEditorConsumer
```

```
[C#]
public class ColumnControlIDEditor.ConsumerImpl : IControlEditorConsumer
```

```
[JScript]
public class ColumnControlIDEditor.ConsumerImpl implements IControlEditorConsumer
```

See Also

[ColumnControlIDEditor.ConsumerImpl Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ColumnControlIDEditor.ConsumerImpl Class Members

[ColumnControlIDEditor.ConsumerImpl overview](#)

Public Constructors

 ColumnControlIDEditor.ConsumerImpl Constructor	
---	--

See Also

[ColumnControlIDEditor.ConsumerImpl Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ColumnDataType Class

Type converter for column's DataType

For a list of all members of this type, see [ColumnDataType members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.ColumnDataType

Syntax

```
[Visual Basic]  
Public Class ColumnDataType
```

```
[C#]  
public class ColumnDataType
```

```
[JScript]  
public class ColumnDataType
```


See Also

[ColumnDataType Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ColumnDataType Class Members

[ColumnDataType overview](#)






Public Constructors

 ColumnDataType Constructor	Overloaded.
---	-------------

Public Properties

 DataType	Converted DataType
---	--------------------

Public Methods

  FromString	
  op_Implicit	Overloaded.
 ToString	Overridden. Returns a string representation of an ColumnDataType object.

See Also

[ColumnDataType Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ColumnEventArgs Class

Event arguments that are passed to event handlers that involve changes to columns.

For a list of all members of this type, see [ColumnEventArgs members](#).

Object Model

ColumnEventArgs

↳ **Column (UltraGridColumn)**

Inheritance Hierarchy

[System.Object](#)

↳ [System.EventArgs](#)

↳ [Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)

↳ **Infragistics.WebUI.UltraWebGrid.ColumnEventArgs**

Syntax

```
[Visual Basic]
Public Class ColumnEventArgs
    Inherits UltraGridEventArgs
```

```
[C#]
public class ColumnEventArgs : UltraGridEventArgs
```

```
[JScript]
public class ColumnEventArgs extends UltraGridEventArgs
```




See Also

[ColumnEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ColumnEventArgs Class Members

[ColumnEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 Column	Returns the Column object associated with the event.
 Data	Returns the Data object associated with the event.

See Also

[ColumnEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ColumnsCollection Class

The Columns collection manages the set of UltraGridColumn objects that correspond to a Band within the grid. For a list of all members of this type, see [ColumnsCollection members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

[System.Collections.CollectionBase](#)

[Infragistics.WebUI.UltraWebGrid.KeyedObjectCollectionBase](#)

[Infragistics.WebUI.UltraWebGrid.ColumnsCollection](#)

[Infragistics.WebUI.UltraWebGrid.SortedColsCollection](#)

Syntax

```
[Visual Basic]
Public Class ColumnsCollection
    Inherits KeyedObjectCollectionBase
    Implements IList, ICollection, IEnumerable, IStateManager, ISupportRollback
```

```
[C#]
public class ColumnsCollection : KeyedObjectCollectionBase,
    IList, ICollection, IEnumerable, IStateManager, ISupportRollback
```

```
[JScript]
public class ColumnsCollection extends KeyedObjectCollectionBase implements
    IList, ICollection, IEnumerable, IStateManager, ISupportRollback
```

Remarks

The top-level UltraWebGrid object exposes the Columns collection of Band(0) and is useful for non-hierarchical, flat grids. Each UltraGridColumn object also contains a Columns collection.


See Also

[ColumnsCollection Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)







ColumnsCollection Class Members

[ColumnsCollection overview](#)















Public Constructors

 ColumnsCollection Constructor	The default constructor for the Columns collection
--	--



Public Properties

 All (Inherited from KeyedObjectCollectionBase)	The collection as an array of objects
 Band	Provides access to the UltraGridColumn object that contains this Columns collection
 Count (Inherited from System.Collections.CollectionBase)	
 IsReadOnly	Overridden. Returns a Boolean value indicating whether the collection is read-only.
 Item	Property indexer for the collection. Uses numeric index.
 Owner (Inherited from KeyedObjectCollectionBase)	Provides public access to the owning object of this collection
















Public Methods

 Add	Overloaded.
 Clear	Removes all UltraGridColumn objects from the collection.
 Contains (Inherited from KeyedObjectCollectionBase)	Returns true if the collection contains this item
 CopyFrom	Populates a Columns collection with all of the UltraGridColumn objects in a different Columns collection.
 CopyTo (Inherited from KeyedObjectCollectionBase)	Copies the items into the array
 Exists (Inherited from KeyedObjectCollectionBase)	Returns true if an object with this key is already in the collection. Note, if key is null or a zero length string this method returns false
 FromKey	Key string indexer for the collection. Uses key string to return a column.
 GetEnumerator	Returns an enumerator object for the Columns collection.
 GetItem (Inherited from KeyedObjectCollectionBase)	Overloaded.
 IndexOf (Inherited from KeyedObjectCollectionBase)	Overloaded.
 Insert	Overloaded.
 Remove	Overloaded.
 RemoveAt	Overloaded.
 ValidateKeyDoesNotExist (Inherited from KeyedObjectCollectionBase)	Overloaded.

Protected Properties

 InnerList (Inherited from System.Collections.CollectionBase)	
 List (Inherited from System.Collections.CollectionBase)	

Protected Methods

 CreateArray (Inherited from KeyedObjectCollectionBase)	Virtual method used by the All 'get' method to create the array it returns.
 InternalAdd (Inherited from KeyedObjectCollectionBase)	Appends the object to the collection
 InternalClear (Inherited from KeyedObjectCollectionBase)	Clears the collection
 InternalInsert (Inherited from KeyedObjectCollectionBase)	Inserts an object into the collection
 InternalRemove (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 InternalRemoveAt (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 OnClear (Inherited from System.Collections.CollectionBase)	
 OnClearComplete (Inherited from System.Collections.CollectionBase)	
 OnInsert (Inherited from System.Collections.CollectionBase)	
 OnInsertComplete (Inherited from System.Collections.CollectionBase)	
 OnRemove (Inherited from System.Collections.CollectionBase)	
 OnRemoveComplete (Inherited from System.Collections.CollectionBase)	
 OnSet (Inherited from System.Collections.CollectionBase)	
 OnSetComplete (Inherited from System.Collections.CollectionBase)	
 OnValidate (Inherited from System.Collections.CollectionBase)	

See Also

[ColumnsCollection Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

EditorControlIDeitorBase Class

General-purpose implementation of an Editor for choosing an **IProvidesEmbeddableEditor** control current residing on this Web Form for use as an Editor of any arbitrary object, component or control.

For a list of all members of this type, see [EditorControlIDeitorBase members](#).

Inheritance Hierarchy

[System.Object](#)

[System.Drawing.Design.UITypeEditor](#)

Infragistics.WebUI.UltraWebGrid.EditorControlIDeitorBase

[Infragistics.WebUI.UltraWebGrid.ColumnControlIDeitor](#)

Syntax

```
[Visual Basic]
Public Class EditorControlIDeitorBase
    Inherits UITypeEditor
```

```
[C#]
public class EditorControlIDeitorBase : UITypeEditor
```

```
[JScript]
public class EditorControlIDeitorBase extends UITypeEditor
```

See Also

[EditorControlIDeitorBase Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)





EditorControlIDeitorBase Class Members

[EditorControlIDeitorBase overview](#)

Public Constructors

 EditorControlIDeitorBase Constructor	
---	--

Public Methods

 EditValue	Overloaded. Overridden.
 GetEditStyle	Overloaded. Overridden.
 GetPaintValueSupported	Overloaded. Overridden.
 PaintValue (Inherited from System.Drawing.Design.UITypeEditor)	Overloaded.

See Also

[EditorControlIDeitorBase Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ExpandEffects Class

This object encapsulates the Internet Explorer Transitions functionality that UltraWebGrid's expandable objects expose.

For a list of all members of this type, see [ExpandEffects members](#).

Inheritance Hierarchy

[System.Object](#)

[Infragistics.WebUI.Shared.WebComponentBase](#)

Infragistics.WebUI.UltraWebGrid.ExpandEffects

Syntax

```
[Visual Basic]
Public Class ExpandEffects
    Inherits WebComponentBase
    Implements IStateManager, ISupportRollback
```

```
[C#]
public class ExpandEffects : WebComponentBase, IStateManager, ISupportRollback
```

```
[JScript]
public class ExpandEffects extends WebComponentBase implements
IStateManager, ISupportRollback
```


See Also

[ExpandEffects Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)









ExpandEffects Class Members

[ExpandEffects overview](#)










Public Constructors

 ExpandEffects Constructor	Returns a reference to the ExpandEffects object for the Grid control.
--	---

Public Properties

 Delay	Specifies the amount of time elapsed before an expandable object will open up in response to a mouse click or a mouseover.
 Duration	Specifies the amount of time elapsed in milliseconds between the beginning and the end of a transition effect.
 HasChanges (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 IsTrackingViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 Opacity	Specifies the amount of transparency applied to the expandable object after the expand effect is complete.
 ShadowColor	Specifies the color of the drop shadow that will appear under the expandable object.
 ShadowWidth	Specifies the width of the drop shadow that will appear under the expandable object.
 Type	

Public Methods

 Commit (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 CopyFrom	Applies the attributes of an existing ExpandEffects object to the current ExpandEffects object, using the property categories specified.
 CreateBackup (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 LoadViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 Reset	Overridden. Resets all properties of the ExpandEffects class to their default values.
 Rollback (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 SaveViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 ToString	Overridden. Returns a string representation of the ExpandEffects object.
 TrackViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	

Protected Properties



[ViewState](#) (Inherited from **Infragistics.WebUI.Shared.WebComponentBase**)

See Also

[ExpandEffects Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

FooterEventArgs Class

Event arguments that are passed to event handlers that involve footers.

For a list of all members of this type, see [FooterEventArgs members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)
[System.EventArgs](#)
[Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)
Infragistics.WebUI.UltraWebGrid.FooterEventArgs

Syntax

```
[Visual Basic]
Public Class FooterEventArgs
    Inherits UltraGridEventArgs
```

```
[C#]
public class FooterEventArgs : UltraGridEventArgs
```

```
[JScript]
public class FooterEventArgs extends UltraGridEventArgs
```



See Also

[FooterEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

FooterEventArgs Class Members

[FooterEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 Rows	A reference to the rows collection that the event applies to. The rows collection forms a data island to which the footer is being applied.

See Also

[FooterEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

FooterItem Class

The FooterItem object is used to implement footer templates in a templated column. The FooterItem functions as the container into which the column footer template is instantiated.

For a list of all members of this type, see [FooterItem members](#).

Object Model

FooterItem

↳ **Column (UltraGridColumn)**

Inheritance Hierarchy

[System.Object](#)

[System.Web.UI.Control](#)

Infragistics.WebUI.UltraWebGrid.FooterItem

Syntax

```
[Visual Basic]
Public Class FooterItem
    Inherits Control
    Implements
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, INamingContainer
```

```
[C#]
public class FooterItem : Control,
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, INamingContainer
```

```
[JScript]
public class FooterItem extends Control implements
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, INamingContainer
```

Remarks

See Also

[FooterItem Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#) | [HeaderItem](#)
















FooterItem Class Members

[FooterItem overview](#)



Public Constructors






 FooterItem Constructor	
---	--

Public Properties







 BindingContainer (Inherited from System.Web.UI.Control)	
 ClientID (Inherited from System.Web.UI.Control)	Gets the server control identifier generated by ASP.NET.
 Column	Returns the column that contains the item template. This property is read-only.
 Controls (Inherited from System.Web.UI.Control)	Gets a System.Web.UI.ControlCollection object that represents the child controls for a specified server control in the UI hierarchy.
 EnableViewState (Inherited from System.Web.UI.Control)	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client.
 FooterSummary	Returns or sets the summary text that will be displayed in the column footer.
 FooterText	Returns the label text that will be displayed in the column footer. This property is read-only.
 ID (Inherited from System.Web.UI.Control)	Gets or sets the programmatic identifier assigned to the server control.
 NamingContainer (Inherited from System.Web.UI.Control)	Gets a reference to the server control's naming container, which creates a unique namespace for differentiating between server controls with the same System.Web.UI.Control.ID property value.
 Page (Inherited from System.Web.UI.Control)	Gets a reference to the System.Web.UI.Page instance that contains the server control.
 Parent (Inherited from System.Web.UI.Control)	Gets a reference to the server control's parent control in the page control hierarchy.
 Site (Inherited from System.Web.UI.Control)	Gets information about the Web site to which the server control belongs.
 TemplateSourceDirectory (Inherited from System.Web.UI.Control)	Gets the virtual directory of the System.Web.UI.Page or System.Web.UI.UserControl that contains the current server control.
 UniqueID (Inherited from System.Web.UI.Control)	Gets the unique, hierarchically-qualified identifier for the server control.
 Visible (Inherited from System.Web.UI.Control)	Gets or sets a value that indicates whether a server control is rendered as UI on the page.

Public Methods








 DataBind (Inherited from System.Web.UI.Control)	Binds a data source to the invoked server control and all its child controls.
 Dispose (Inherited from System.Web.UI.Control)	Enables a server control to perform final clean up before it is released from memory.

 FindControl (Inherited from System.Web.UI.Control)	Overloaded.
 HasControls (Inherited from System.Web.UI.Control)	Determines if the server control contains any child controls.
 RenderControl (Inherited from System.Web.UI.Control)	Outputs server control content to a provided System.Web.UI.HtmlTextWriter object and stores tracing information about the control if tracing is enabled.
 ResolveUrl (Inherited from System.Web.UI.Control)	Converts a URL into one that is usable on the requesting client.
 SetRenderMethodDelegate (Inherited from System.Web.UI.Control)	




Public Events






 DataBinding (Inherited from System.Web.UI.Control)	
 Disposed (Inherited from System.Web.UI.Control)	
 Init (Inherited from System.Web.UI.Control)	
 Load (Inherited from System.Web.UI.Control)	
 PreRender (Inherited from System.Web.UI.Control)	
 Unload (Inherited from System.Web.UI.Control)	

Protected Properties

 ChildControlsCreated (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control's child controls have been created.
 Context (Inherited from System.Web.UI.Control)	Gets the System.Web.HttpContext object associated with the server control for the current Web request.
 Events (Inherited from System.Web.UI.Control)	Gets a list of event handler delegates for the control. This property is read-only.
 HasChildViewState (Inherited from System.Web.UI.Control)	Gets a value indicating whether the current server control's child controls have any saved view-state settings.
 IsTrackingViewState (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control is saving changes to its view state.
 ViewState (Inherited from System.Web.UI.Control)	Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page.
 ViewStateIgnoresCase (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the System.Web.UI.StateBag object is case-insensitive.

Protected Methods

 AddedControl (Inherited from System.Web.UI.Control)	Called after a control is added to the System.Web.UI.Control.Controls collection of another control.
 AddParsedSubObject (Inherited from System.Web.UI.Control)	Notifies the server control that an element, either XML or HTML, was parsed, and adds the element to the server control's System.Web.UI.ControlCollection object.
 BuildProfileTree (Inherited from System.Web.UI.Control)	

 ClearChildViewState (Inherited from System.Web.UI.Control)	Deletes the view-state information for all the server control's child controls.
 CreateChildControls (Inherited from System.Web.UI.Control)	Notifies server controls that use composition-based implementation to create any child controls they contain in preparation for posting back or rendering.
 CreateControlCollection (Inherited from System.Web.UI.Control)	Creates a new System.Web.UI.ControlCollection object to hold the child controls (both literal and server) of the server control.
 EnsureChildControls (Inherited from System.Web.UI.Control)	Determines whether the server control contains child controls. If it does not, it creates child controls.
 IsLiteralContent (Inherited from System.Web.UI.Control)	Determines if the server control holds only literal content.
 LoadViewState (Inherited from System.Web.UI.Control)	Restores view-state information from a previous page request that was saved by the System.Web.UI.Control.SaveViewState method.
 MapPathSecure (Inherited from System.Web.UI.Control)	Retrieves the physical path that a virtual path, either absolute or relative, maps to.
 OnBubbleEvent (Inherited from System.Web.UI.Control)	Determines whether the event for the server control is passed up the page's UI server control hierarchy.
 OnDataBinding (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.DataBinding event.
 OnInit (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Init event.
 OnLoad (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Load event.
 OnPreRender (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.PreRender event.
 OnUnload (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Unload event. Server controls should perform any final cleanup, such as closing files, closing database connections, and discarding objects, during this stage of the server control lifecycle.
 RaiseBubbleEvent (Inherited from System.Web.UI.Control)	Assigns any sources of the event and its information to the control's parent.
 RemovedControl (Inherited from System.Web.UI.Control)	Called after a control is removed from the System.Web.UI.Control.Controls collection of another control.
 Render (Inherited from System.Web.UI.Control)	Sends server control content to a provided System.Web.UI.HtmlTextWriter object, which writes the content to be rendered on the client.
 RenderChildren (Inherited from System.Web.UI.Control)	Outputs the content of a server control's children to a provided System.Web.UI.HtmlTextWriter object, which writes the content to be rendered on the client.
 SaveViewState (Inherited from System.Web.UI.Control)	Saves any server control view-state changes that have occurred since the time the page was posted back to the server.
 TrackViewState (Inherited from System.Web.UI.Control)	Causes tracking of view-state changes to the server control so they can be stored in the server control's System.Web.UI.StateBag object. This object is accessible through the System.Web.UI.Control.ViewState property.

See Also

[FooterItem Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#) | [HeaderItem](#)

GridItemStyle Class

The GridItemStyle class handles properties and methods directly related to the appearance of an object inside the grid. GridItemStyles are applied to the cells, footers and column headers of the grid.

For a list of all members of this type, see [GridItemStyle members](#).

Inheritance Hierarchy

[System.Object](#)
[System.MarshalByRefObject](#)
[System.ComponentModel.Component](#)
[System.Web.UI.WebControls.Style](#)
[Infragistics.WebUI.Shared.Style](#)
Infragistics.WebUI.UltraWebGrid.GridItemStyle

Syntax

```
[Visual Basic]
Public Class GridItemStyle
    Inherits Style
    Implements IComponent, IDisposable, IStateManager, ISupportRollback
```

```
[C#]
public class GridItemStyle : Style,
IComponent, IDisposable, IStateManager, ISupportRollback
```

```
[JScript]
public class GridItemStyle extends Style implements
IComponent, IDisposable, IStateManager, ISupportRollback
```


See Also

[GridItemStyle Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)













GridItemStyle Class Members







[GridItemStyle overview](#)

Public Constructors

 GridItemStyle Constructor	Overloaded.
--	-------------

Public Properties

 BackColor (Inherited from System.Web.UI.WebControls.Style)	Gets or sets the background color of the Web server control.
 BackgroundImage (Inherited from Infragistics.WebUI.Shared.Style)	
 BorderColor (Inherited from System.Web.UI.WebControls.Style)	Gets or sets the border color of the Web server control.
 BorderDetails (Inherited from Infragistics.WebUI.Shared.Style)	
 BorderStyle (Inherited from System.Web.UI.WebControls.Style)	Gets or sets the border style of the Web server control.
 BorderWidth (Inherited from System.Web.UI.WebControls.Style)	Gets or sets the border width of the Web server control.
 Container (Inherited from System.ComponentModel.Component)	Gets the System.ComponentModel.IContainer that contains the System.ComponentModel.Component .
 CssClass (Inherited from System.Web.UI.WebControls.Style)	Gets or sets the CSS class rendered by the Web server control on the client.
 Cursor (Inherited from Infragistics.WebUI.Shared.Style)	
 CustomRules (Inherited from Infragistics.WebUI.Shared.Style)	
 Font (Inherited from System.Web.UI.WebControls.Style)	Gets the font properties associated with the Web server control.
 ForeColor (Inherited from System.Web.UI.WebControls.Style)	Gets or sets the foreground color (typically the color of the text) of the Web server control.
 HasBorderDetails (Inherited from Infragistics.WebUI.Shared.Style)	
 HasChanges (Inherited from Infragistics.WebUI.Shared.Style)	
 HasMargin (Inherited from Infragistics.WebUI.Shared.Style)	
 HasPadding (Inherited from Infragistics.WebUI.Shared.Style)	
 Height (Inherited from System.Web.UI.WebControls.Style)	Gets or sets the height of the Web server control.
 HorizontalAlign	Returns or sets a value that specifies the horizontal alignment of the text associated with a GridItemStyle object.
 Margin (Inherited from Infragistics.WebUI.Shared.Style)	

 Padding (Inherited from Infragistics.WebUI.Shared.Style)	
 Site (Inherited from System.ComponentModel.Component)	Gets or sets the System.ComponentModel.Site of the System.ComponentModel.Component .
 TextOverflow	Defines text's behavior in a nowrap-mode (Internet Explorer 6 and above only).
 VerticalAlign	Returns or sets a value that specifies the vertical alignment of the text associated with a GridItemStyle object.
 Width (Inherited from System.Web.UI.WebControls.Style)	Gets or sets the width of the Web server control.
 Wrap	Returns or sets a Boolean value that determines whether the text associated with a GridItemStyle object is wrapped.

Public Methods






 AddAttributesToRender (Inherited from System.Web.UI.WebControls.Style)	Overloaded.
 Commit (Inherited from Infragistics.WebUI.Shared.Style)	
 CopyFrom	Overridden. Duplicates the style properties of the specified Style into the instance of the Style class from which this method is invoked.
 CreateBackup (Inherited from Infragistics.WebUI.Shared.Style)	
 CreateObjRef (Inherited from System.MarshalByRefObject)	
 Dispose (Inherited from System.ComponentModel.Component)	Overloaded. Releases all resources used by the System.ComponentModel.Component .
 GetLifetimeService (Inherited from System.MarshalByRefObject)	
 InitializeLifetimeService (Inherited from System.MarshalByRefObject)	
 IsEmpty	Overloaded. Overridden. Returns a Boolean value that determines whether a GridItemStyle object is set to Nothing or Null.
 IsFontEmpty (Inherited from Infragistics.WebUI.Shared.Style)	
 MergeWith	Overloaded.
 Reset	Overloaded. Overridden. Resets all properties of the GridItemStyle class to their default values.
 ResetBorderDetails (Inherited from Infragistics.WebUI.Shared.Style)	
 ResetMargin (Inherited from Infragistics.WebUI.Shared.Style)	
 ResetPadding (Inherited from Infragistics.WebUI.Shared.Style)	
 Rollback (Inherited from Infragistics.WebUI.Shared.Style)	
 SetDesignerOwner (Inherited from Infragistics.WebUI.Shared.Style)	

 ToString (Inherited from Infragistics.WebUI.Shared.Style)	
---	--











Public Events

 Disposed (Inherited from System.ComponentModel.Component)	
--	--

Protected Properties

 DesignMode (Inherited from System.ComponentModel.Component)	Gets a value that indicates whether the System.ComponentModel.Component is currently in design mode.
 Events (Inherited from System.ComponentModel.Component)	Gets the list of event handlers that are attached to this System.ComponentModel.Component .
 IsEmpty (Inherited from System.Web.UI.WebControls.Style)	
 IsTrackingViewState (Inherited from System.Web.UI.WebControls.Style)	Returns a value indicating whether any style elements have been defined in the state bag.
 ViewState (Inherited from System.Web.UI.WebControls.Style)	

Protected Methods

 Finalize (Inherited from System.ComponentModel.Component)	Releases unmanaged resources and performs other cleanup operations before the System.ComponentModel.Component is reclaimed by garbage collection.
 GetService (Inherited from System.ComponentModel.Component)	Returns an object that represents a service provided by the System.ComponentModel.Component or by its System.ComponentModel.Container .
 HasData	Overridden.
 LoadViewState (Inherited from System.Web.UI.WebControls.Style)	
 SaveViewState (Inherited from System.Web.UI.WebControls.Style)	
 SetBit (Inherited from System.Web.UI.WebControls.Style)	
 ShouldSerializeBorderDetails (Inherited from Infragistics.WebUI.Shared.Style)	
 ShouldSerializeMargin (Inherited from Infragistics.WebUI.Shared.Style)	
 ShouldSerializePadding (Inherited from Infragistics.WebUI.Shared.Style)	
 TrackViewState (Inherited from System.Web.UI.WebControls.Style)	

See Also

[GridItemStyle Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

GroupByBox Class

The GroupByBox object represents the area at the top of the grid that is used for dragging or clicking column headers in order to group rows by the common values of that column.

For a list of all members of this type, see [GroupByBox members](#).

Object Model

GroupByBox

BandLabelStyle (GridItemStyle)

Style (GridItemStyle)

Inheritance Hierarchy

[System.Object](#)

[Infragistics.WebUI.Shared.WebComponentBase](#)

[Infragistics.WebUI.UltraWebGrid.SpecialBoxBase](#)

Infragistics.WebUI.UltraWebGrid.GroupByBox

Syntax

```
[Visual Basic]
Public Class GroupByBox
    Inherits SpecialBoxBase
    Implements IStateManager, ISupportRollback
```

```
[C#]
public class GroupByBox : SpecialBoxBase, IStateManager, ISupportRollback
```

```
[JScript]
public class GroupByBox extends SpecialBoxBase implements IStateManager, ISupportRollback
```


See Also

[GroupByBox Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)












GroupBox Class Members

[GroupBox overview](#)




Public Constructors

 GroupBox Constructor	
---	--


Public Properties

 BandLabelStyle	Returns a reference to or sets a GridItemStyle object that determines how the GroupBox band labels are rendered on the client.
 ButtonConnectorColor (Inherited from SpecialBoxBase)	Returns a reference to or sets a Color object that specifies the color of the button connector lines for an object.
 ButtonConnectorStyle (Inherited from SpecialBoxBase)	Returns a reference to or sets a BorderStyle object that specifies the style of the button connector lines for an object.
 HasBandLabelStyle	Returns a Boolean value that determines whether the BandLabelStyle property is currently set to a GridItemStyle object.
 HasStyle (Inherited from SpecialBoxBase)	Returns a Boolean value that determines whether the Style property is currently set to a GridItemStyle object.
 Hidden	Returns or sets a Boolean value that determines whether the GroupBox box is hidden.
 IsTrackingViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 Location	Returns or sets a value that specifies where the GroupBox band labels are rendered to the client.
 Prompt	Returns or sets a string of text that indicates to the user how to perform GroupBy operations.
 ShowBandLabels	Returns or sets a value that specifies the style of the type of band levels shown in the GroupByBox.
 Style (Inherited from SpecialBoxBase)	Returns a reference to or sets a GridItemStyle object that determines how a button object is rendered on the client.

Public Methods

 CopyFrom	Overridden. Duplicates the properties of the specified SpecialBoxBase into the instance of the GroupBox class from which this method is invoked.
 Reset	Overridden. Resets all properties of the GroupBox class to their default values.
 ToString	Overridden. Returns a string representation of a GroupBox object.

Protected Properties

 ViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
--	--

See Also

[GroupByBox Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

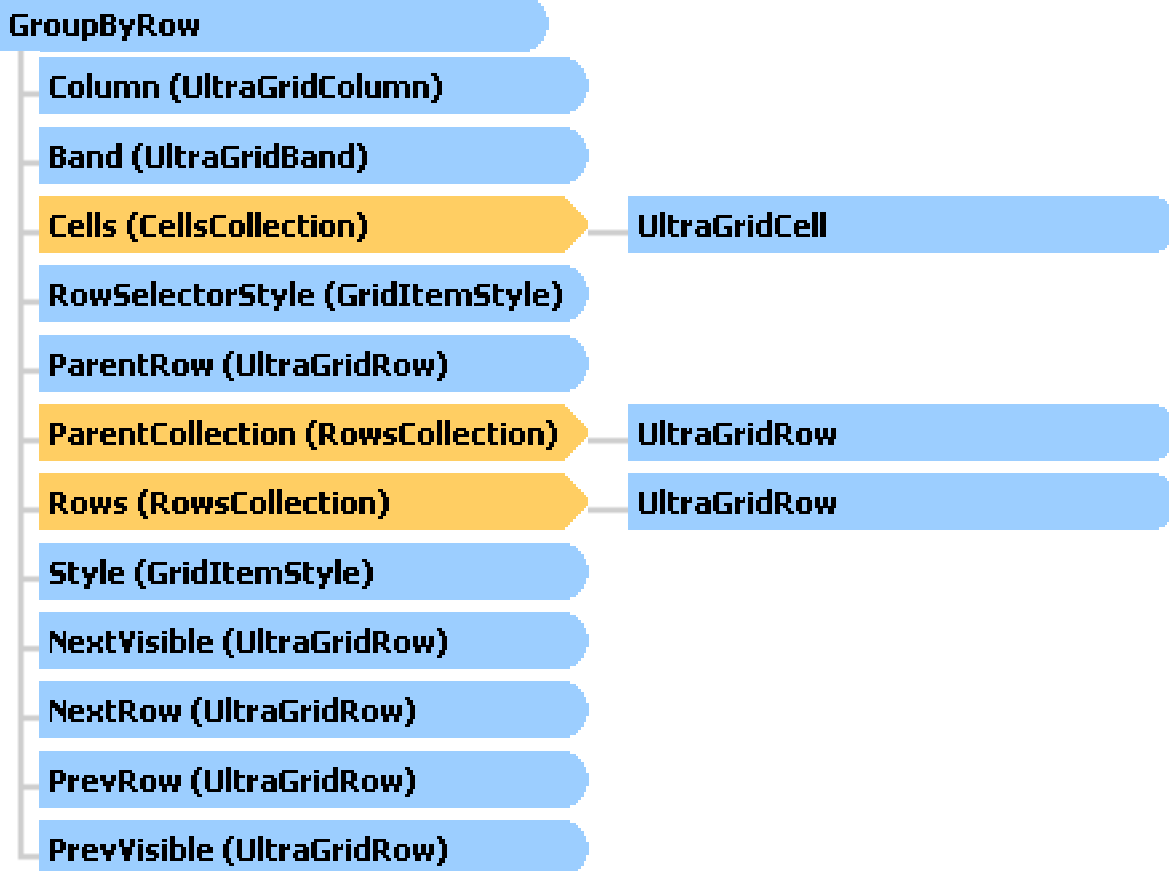
© 2004 • Infragistics, Inc.

GroupByRow Class

The GroupByRow object is a row that represents a grouped column. All rows in the data source that have common/equal values in the column that is grouped will be children of the GroupByRow object. The GroupByRow object contains a reference to the grouped column as well as a reference to the particular value that the child rows have in common.

For a list of all members of this type, see [GroupByRow members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)
 Infragistics.WebUI.UltraWebGrid.KeyedObjectBase
[Infragistics.WebUI.UltraWebGrid.UltraGridRow](#)
Infragistics.WebUI.UltraWebGrid.GroupByRow

Syntax

```
[Visual Basic]
Public Class GroupByRow
    Inherits UltraGridRow
    Implements IKeyedObject, IComparable, IStateManager
```

```
[C#]
public class GroupByRow : UltraGridRow, IKeyedObject, IComparable, IStateManager
```

```
[JScript]
public class GroupByRow extends UltraGridRow implements
IKeyedObject, IComparable, IStateManager
```


See Also

[GroupByRow Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)























GroupByRow Class Members

[GroupByRow overview](#)

Public Constructors











 GroupByRow Constructor	Overloaded.
---	-------------












Public Properties

 Activated (Inherited from UltraGridRow)	Indicates whether this row has been activated.
 Band (Inherited from UltraGridRow)	Returns the UltraGridBand object of the band that contains the row.
 Cells (Inherited from UltraGridRow)	Returns a collection of the UltraGridCell objects that make up the row.
 Column	Returns a reference to or sets the UltraGridColumn object that specifies the associated column in a GroupBy operation.
 DataChanged (Inherited from UltraGridRow)	Indicates whether the row's data has been edited or this is a new or deleted row and the changes did not yet committed to the database
 DataKey (Inherited from UltraGridRow)	Returns the key value for the row.
 DataSourceIndex (Inherited from UltraGridRow)	Specifies the index (position) in the recordsource of the data record that corresponds to the current row.
 Expanded (Inherited from UltraGridRow)	Returns the expanded state of the row.
 HasCells (Inherited from UltraGridRow)	Determines whether the row contains any cells.
 HasChildRows (Inherited from UltraGridRow)	Determines whether the row has a set of child rows.
 HasNextSibling (Inherited from UltraGridRow)	Determines whether a row has a sibling row below it.
 HasParent (Inherited from UltraGridRow)	Determines whether the row has a parent row or if it is the top-level row.
 HasPrevSibling (Inherited from UltraGridRow)	Determines whether a row has a sibling row above it.
 HasRowSelectorStyle (Inherited from UltraGridRow)	Determines if a style has been applied to the row selectors at the row level.
 HasStyle (Inherited from UltraGridRow)	Determines if a style has been applied to the row at the row level.
 Height (Inherited from UltraGridRow)	The height of the row, in pixels.
 HeightResolved (Inherited from UltraGridRow)	The height of the row, in pixels. This property will always return the setting that is in control of the row.
 Hidden (Inherited from UltraGridRow)	Determines whether the object will be displayed.
 Index (Inherited from UltraGridRow)	Returns the index of the row object in the UltraGridRows collection
 Key (Inherited from UltraGridRow)	Returns or sets a unique string identifier for this Row within the UltraGridRows collection.
 Level (Inherited from UltraGridRow)	Returns the number of the band containing the row within the grid's data hierarchy. This property is read-only.
 NextRow (Inherited from UltraGridRow)	Returns the next sibling row in the band.
 NextVisible (Inherited from UltraGridRow)	Returns the next visible sibling row in the band.


 ParentCollection (Inherited from UltraGridRow)	Returns the Rows collection that contains the current row. You can use this collection to enumerate the siblings of the current row.
 ParentRow (Inherited from UltraGridRow)	Gets the row immediately above the current one in the data hierarchy, if any.
 PrevRow (Inherited from UltraGridRow)	Returns the previous sibling row in the band.
 PrevVisible (Inherited from UltraGridRow)	Returns the previous visible sibling row in the band.
 Rows (Inherited from UltraGridRow)	Returns the collection of the current row's child rows, if any.
 RowSelectorStyle (Inherited from UltraGridRow)	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the row selector.
 Selected (Inherited from UltraGridRow)	Determines the selected state of the row.
 ShowExpand (Inherited from UltraGridRow)	Returns or sets a Boolean value that determines whether a Row object is displayed with an expand image next to it, regardless of whether or not it has children. This property is only effective in conjunction with the use of the LoadOnDemand property of the DisplayLayout object.
 Sizing (Inherited from UltraGridRow)	Determines whether the row height can be changed.
 SizingResolved (Inherited from UltraGridRow)	Determines whether the row height can be changed. This property will always return the setting that is in control of the row.
 Style (Inherited from UltraGridRow)	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the row.
 Tag (Inherited from UltraGridRow)	A field for storing user-defined, object-related information.
 Text	Returns or sets the group by row description. If this property is null then default text, based on GroupByRowDescriptionMask, is used.
 Value	Returns or sets the cell value that is common to all of the rows being grouped in a GroupBy operation.

Public Methods






 Activate (Inherited from UltraGridRow)	Sets the current row as the active row.
 Collapse (Inherited from UltraGridRow)	Collapses the children of the current row.
 CompareTo (Inherited from UltraGridRow)	Compares the row to the sorted data in the band and returns the row's position within the sort order.
 CopyFrom (Inherited from UltraGridRow)	Copy the UltraGridRow object
 Delete (Inherited from UltraGridRow)	Deletes the current row.
 Expand (Inherited from UltraGridRow)	Expand the children of the current row.
 ExpandAncestors (Inherited from UltraGridRow)	Expands all ancestor rows of the current row.
 Find (Inherited from UltraGridRow)	Overloaded. Searches for a cell in the row, which value starts with provided string. Case is ignored. Uses previously passed parameters.
 FindNext (Inherited from UltraGridRow)	Overloaded. Continues search from previously found cell.
 GetCellText (Inherited from UltraGridRow)	Returns string representation of the value of the specified cell.

 GetCellValue (Inherited from UltraGridRow)	Returns value of the cell in its native form.
 IsActiveRow (Inherited from UltraGridRow)	Determines whether the row is the active row.
 IsAlternate (Inherited from UltraGridRow)	Determines whether the row is an alternate (even-numbered) row.
 IsChild (Inherited from UltraGridRow)	Specifies whether the passed-in row is the direct child of the current row.
 IsDescendant (Inherited from UltraGridRow)	Specifies whether the passed-in row is a descendant of the current row.
 IsExpanded (Inherited from UltraGridRow)	Determines whether the row is currently expanded.
 IsSelectable (Inherited from UltraGridRow)	Determines whether the row may be selected, based on the current selection settings of the grid and the band.
 ResetKey (Inherited from KeyedObjectBase)	Sets the key back to null
 ShouldSerializeKey (Inherited from KeyedObjectBase)	Returns true if the key needs to be serialized (not null)
 Toggle (Inherited from UltraGridRow)	Obtain the UltraWebGrid object to which the row is attached. Changes the expanded state of the row to Collapsed from Expanded or to Expanded from Collapsed
 ToString (Inherited from UltraGridRow)	Returns a string representation of an UltraGridRow object.

Protected Properties

 ViewState (Inherited from UltraGridRow)	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.
--	--

Protected Methods

 LoadViewState (Inherited from UltraGridRow)	
 OnAddedToCollection (Inherited from UltraGridRow)	
 OnRemovedFromCollection (Inherited from KeyedObjectBase)	Called when this object is being removed from the passed in collection. The default implementation nulls out the primaryCollection if the passed in collection matches
 SaveViewState (Inherited from UltraGridRow)	
 TrackViewState (Inherited from UltraGridRow)	

See Also

[GroupByRow Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

HeaderItem Class

The HeaderItem object is used to implement header templates in a templated column. The HeaderItem functions as the container into which the column header template is instantiated.

For a list of all members of this type, see [HeaderItem members](#).

Object Model

HeaderItem

Column (UltraGridColumn)

Inheritance Hierarchy

[System.Object](#)

[System.Web.UI.Control](#)

Infragistics.WebUI.UltraWebGrid.HeaderItem

Syntax

```
[Visual Basic]
Public Class HeaderItem
    Inherits Control
    Implements
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, INamingContainer
```

```
[C#]
public class HeaderItem : Control,
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, INamingContainer
```

```
[JScript]
public class HeaderItem extends Control implements
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, INamingContainer
```

Remarks

See Also

[HeaderItem Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#) | [FooterItem](#)
















HeaderItem Class Members

[HeaderItem overview](#)



Public Constructors






 HeaderItem Constructor	
---	--

Public Properties







 BindingContainer (Inherited from System.Web.UI.Control)	
 ClientID (Inherited from System.Web.UI.Control)	Gets the server control identifier generated by ASP.NET.
 Column	Returns the column that contains the item template. This property is read-only.
 Controls (Inherited from System.Web.UI.Control)	Gets a System.Web.UI.ControlCollection object that represents the child controls for a specified server control in the UI hierarchy.
 EnableViewState (Inherited from System.Web.UI.Control)	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client.
 HeaderText	Returns a string value containing the header text that will be displayed for the column. This property is read-only.
 ID (Inherited from System.Web.UI.Control)	Gets or sets the programmatic identifier assigned to the server control.
 NamingContainer (Inherited from System.Web.UI.Control)	Gets a reference to the server control's naming container, which creates a unique namespace for differentiating between server controls with the same System.Web.UI.Control.ID property value.
 Page (Inherited from System.Web.UI.Control)	Gets a reference to the System.Web.UI.Page instance that contains the server control.
 Parent (Inherited from System.Web.UI.Control)	Gets a reference to the server control's parent control in the page control hierarchy.
 Site (Inherited from System.Web.UI.Control)	Gets information about the Web site to which the server control belongs.
 SortIndicator	Returns a string value containing the URL of the image file being used as the column's sort indicator. This property is read-only.
 TemplateSourceDirectory (Inherited from System.Web.UI.Control)	Gets the virtual directory of the System.Web.UI.Page or System.Web.UI.UserControl that contains the current server control.
 UniqueID (Inherited from System.Web.UI.Control)	Gets the unique, hierarchically-qualified identifier for the server control.
 Visible (Inherited from System.Web.UI.Control)	Gets or sets a value that indicates whether a server control is rendered as UI on the page.

Public Methods








 DataBind (Inherited from System.Web.UI.Control)	Binds a data source to the invoked server control and all its child controls.
 Dispose (Inherited from System.Web.UI.Control)	Enables a server control to perform final clean up before it is released from memory.

 FindControl (Inherited from System.Web.UI.Control)	Overloaded.
 HasControls (Inherited from System.Web.UI.Control)	Determines if the server control contains any child controls.
 RenderControl (Inherited from System.Web.UI.Control)	Outputs server control content to a provided System.Web.UI.HtmlTextWriter object and stores tracing information about the control if tracing is enabled.
 ResolveUrl (Inherited from System.Web.UI.Control)	Converts a URL into one that is usable on the requesting client.
 SetRenderMethodDelegate (Inherited from System.Web.UI.Control)	




Public Events






 DataBinding (Inherited from System.Web.UI.Control)	
 Disposed (Inherited from System.Web.UI.Control)	
 Init (Inherited from System.Web.UI.Control)	
 Load (Inherited from System.Web.UI.Control)	
 PreRender (Inherited from System.Web.UI.Control)	
 Unload (Inherited from System.Web.UI.Control)	

Protected Properties

 ChildControlsCreated (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control's child controls have been created.
 Context (Inherited from System.Web.UI.Control)	Gets the System.Web.HttpContext object associated with the server control for the current Web request.
 Events (Inherited from System.Web.UI.Control)	Gets a list of event handler delegates for the control. This property is read-only.
 HasChildViewState (Inherited from System.Web.UI.Control)	Gets a value indicating whether the current server control's child controls have any saved view-state settings.
 IsTrackingViewState (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control is saving changes to its view state.
 ViewState (Inherited from System.Web.UI.Control)	Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page.
 ViewStateIgnoresCase (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the System.Web.UI.StateBag object is case-insensitive.

Protected Methods

 AddedControl (Inherited from System.Web.UI.Control)	Called after a control is added to the System.Web.UI.Control.Controls collection of another control.
 AddParsedSubObject (Inherited from System.Web.UI.Control)	Notifies the server control that an element, either XML or HTML, was parsed, and adds the element to the server control's System.Web.UI.ControlCollection object.
 BuildProfileTree (Inherited from System.Web.UI.Control)	

 ClearChildViewState (Inherited from System.Web.UI.Control)	Deletes the view-state information for all the server control's child controls.
 CreateChildControls (Inherited from System.Web.UI.Control)	Notifies server controls that use composition-based implementation to create any child controls they contain in preparation for posting back or rendering.
 CreateControlCollection (Inherited from System.Web.UI.Control)	Creates a new System.Web.UI.ControlCollection object to hold the child controls (both literal and server) of the server control.
 EnsureChildControls (Inherited from System.Web.UI.Control)	Determines whether the server control contains child controls. If it does not, it creates child controls.
 IsLiteralContent (Inherited from System.Web.UI.Control)	Determines if the server control holds only literal content.
 LoadViewState (Inherited from System.Web.UI.Control)	Restores view-state information from a previous page request that was saved by the System.Web.UI.Control.SaveViewState method.
 MapPathSecure (Inherited from System.Web.UI.Control)	Retrieves the physical path that a virtual path, either absolute or relative, maps to.
 OnBubbleEvent (Inherited from System.Web.UI.Control)	Determines whether the event for the server control is passed up the page's UI server control hierarchy.
 OnDataBinding (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.DataBinding event.
 OnInit (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Init event.
 OnLoad (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Load event.
 OnPreRender (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.PreRender event.
 OnUnload (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Unload event. Server controls should perform any final cleanup, such as closing files, closing database connections, and discarding objects, during this stage of the server control lifecycle.
 RaiseBubbleEvent (Inherited from System.Web.UI.Control)	Assigns any sources of the event and its information to the control's parent.
 RemovedControl (Inherited from System.Web.UI.Control)	Called after a control is removed from the System.Web.UI.Control.Controls collection of another control.
 Render (Inherited from System.Web.UI.Control)	Sends server control content to a provided System.Web.UI.HtmlTextWriter object, which writes the content to be rendered on the client.
 RenderChildren (Inherited from System.Web.UI.Control)	Outputs the content of a server control's children to a provided System.Web.UI.HtmlTextWriter object, which writes the content to be rendered on the client.
 SaveViewState (Inherited from System.Web.UI.Control)	Saves any server control view-state changes that have occurred since the time the page was posted back to the server.
 TrackViewState (Inherited from System.Web.UI.Control)	Causes tracking of view-state changes to the server control so they can be stored in the server control's System.Web.UI.StateBag object. This object is accessible through the System.Web.UI.Control.ViewState property.

See Also

[HeaderItem Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#) | [FooterItem](#)

ImageUrls Class

For a list of all members of this type, see [ImageUrls members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.ImageUrls

Syntax

```
[Visual Basic]
Public Class ImageUrls
    Implements IStateManager
```

```
[C#]
public class ImageUrls : IStateManager
```

```
[JScript]
public class ImageUrls implements IStateManager
```

See Also

[ImageUrls Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)
















ImageUrls Class Members

[ImageUrls overview](#)





Public Constructors

 ImageUrls Constructor	
--	--


Public Properties

 BlankImage	The name of the blank image. (Used as a spacer inside of the RowSelector area of the grid).
 CollapseImage	The name of the image used as the "collapse rows" icon.
 CurrentEditRowImage	The name of the selected editable row image.
 CurrentRowImage	The name of the active row image.
 ExpandImage	The name of the image used as the "expand rows" icon.
 GridCornerImage	The name of the image that will appear in the left upper corner of the grid.
 GroupByImage	The name of the group by image. (Used in down-level browser that do not support drag and drop).
 GroupDownArrow	The name of the down arrow image.
 GroupUpArrow	The name of the up arrow image.
 ImageDirectory	The path of the directory where the images used by the control are stored.
 NewRowImage	The name of the new row image.
 RowLabelBlankImage	The name of the image for the row selectors. (Used as a spacer inside of the RowSelector area of the grid).
 SortAscending	The name of the ascending sort image.
 SortDescending	The name of the descending sort image.
 UnGroupByImage	The name of the ungroup image. (Used in down-level browser that do not support drag and drop).

Public Methods

 Clone	Clone the ImageUrls object
 CopyFrom	Copy the ImageUrls object
 Reset	Resets all properties of the ImageUrls class to their default values.
 ToString	Overridden. Returns a string representation of an ImageUrls object.

Protected Properties

 ViewState	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.
--	--

See Also

InGridRenderer Class

Plug-in renderer implementation that allows extensions to render themselves in special docking regions within the UltraWebGrid.

For a list of all members of this type, see [InGridRenderer members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.InGridRenderer

Syntax

```
[Visual Basic]
Public Class InGridRenderer
    Implements IPlugInRender
```

```
[C#]
public class InGridRenderer : IPlugInRender
```

```
[JScript]
public class InGridRendererer implements IPlugInRender
```


See Also

[InGridRenderer Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

InGridRenderer Class Members

[InGridRenderer overview](#)

Public Constructors

 InGridRenderer Constructor	Creates an instance of an In-Grid external renderer.
---	--

See Also

[InGridRenderer Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

InvalidCellsEnumerator Class

Enumerator for invalid cells in a grid.

For a list of all members of this type, see [InvalidCellsEnumerator members](#).

Object Model

InvalidCellsEnumerator

Current (ChangedCellPair)

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.InvalidCellsEnumerator

Syntax

```
[Visual Basic]
Public Class InvalidCellsEnumerator
    Implements IEnumerator
```

```
[C#]
public class InvalidCellsEnumerator : IEnumerator
```

```
[JScript]
public class InvalidCellsEnumerator implements IEnumerator
```

See Also

[InvalidCellsEnumerator Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

InvalidCellsEnumerator Class Members

[InvalidCellsEnumerator overview](#)



Public Constructors

 InvalidCellsEnumerator Constructor	
---	--

Public Properties

 Current	
--	--

Public Methods

 MoveNext	
 Reset	

See Also

[InvalidCellsEnumerator Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

LayoutEventArgs Class

Event arguments that are passed to event handlers that involve layout initialization.

For a list of all members of this type, see [LayoutEventArgs members](#).

Object Model

LayoutEventArgs

└─ **Layout (UltraGridLayout)**

Inheritance Hierarchy

[System.Object](#)

└─ [System.EventArgs](#)

└─ [Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)

└─ **Infragistics.WebUI.UltraWebGrid.LayoutEventArgs**

Syntax

```
[Visual Basic]
Public Class LayoutEventArgs
    Inherits UltraGridEventArgs
```

```
[C#]
public class LayoutEventArgs : UltraGridEventArgs
```

```
[JScript]
public class LayoutEventArgs extends UltraGridEventArgs
```



See Also

[LayoutEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

LayoutEventArgs Class Members

[LayoutEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 Layout	A reference to the DisplayLayout object of UltraWebGrid.

See Also

[LayoutEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

PageEventArgs Class

Event arguments that are passed to event handlers that involve paging.

For a list of all members of this type, see [PageEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)

Infragistics.WebUI.UltraWebGrid.PageEventArgs

Syntax

```
[Visual Basic]
Public Class PageEventArgs
    Inherits UltraGridEventArgs
```

```
[C#]
public class PageEventArgs : UltraGridEventArgs
```

```
[JScript]
public class PageEventArgs extends UltraGridEventArgs
```




See Also

[PageEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

PageEventArgs Class Members

[PageEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 NewPageIndex	The index of the new page that has been selected. This value is the same as the Pager.CurrentPageIndex property.
 OldPageIndex	The index of the page that was previously displayed.

See Also

[PageEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

Pager Class

The Pager class is responsible for controlling the paging behavior and appearance within the UltraWebGrid. There are three types of paging supported: 1. Built-in paging, Standard paging, and Custom paging. Built-in paging is completely automatic and does not require any coding to implement. Built-in paging is automatically enabled when the `DisplayLayout.EnableInternalRowsManagement` property is set to true. With this style of paging, however, there is a large ViewState requirement, as all pages of the query must be available to the grid without the need for a call to `DataBind()`. Standard paging simply requires that the application responds to the `PageIndexChanged` event to set the new page number and call `DataBind()`. UltraWebGrid indexes into the `DataSource` and finds the correct starting point and loads the grid with the `PageSize` number of rows. With Custom paging, it is the application's responsibility to provide a `DataSource` that has the exact set of rows to be displayed. UltraWebGrid will start at the beginning of the `DataSource` and load the `PageSize` number of rows. In this scenario, the `DataSource` query has to be adjusted to return the correct set of rows, based upon the `PageIndex` or some other criteria.

For a list of all members of this type, see [Pager members](#).

Object Model

Pager

Style (GridItemStyle)

Inheritance Hierarchy

[System.Object](#)

[Infragistics.WebUI.Shared.WebComponentBase](#)

Infragistics.WebUI.UltraWebGrid.Pager

Syntax

```
[Visual Basic]
Public Class Pager
    Inherits WebComponentBase
    Implements IStateManager, ISupportRollback
```

```
[C#]
public class Pager : WebComponentBase, IStateManager, ISupportRollback
```

```
[JScript]
public class Pager extends WebComponentBase implements IStateManager, ISupportRollback
```

See Also

[Pager Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)



















Pager Class Members

[Pager overview](#)







Public Constructors

 Pager Constructor	
--	--


Public Properties

 Alignment	Returns or sets a value that specifies the horizontal alignment of the text displayed when paging is enabled.
 AllowCustomPaging	Returns or sets a Boolean value that determines whether custom paging is allowed.
 AllowPaging	Returns or sets a Boolean value that determines whether paging is allowed.
 ChangeLinksColor	Allows to apply the pager fore color to the page links.
 CurrentPageIndex	Returns or sets a value that specifies the current data page displayed by the grid.
 CustomLabels	Returns or sets a list of strings that determine the text to be displayed in lieu of page indices when paging is enabled.
 HasChanges (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 HasStyle	Returns a Boolean value that determines whether the Style property is currently set to a GridItemStyle object.
 IsTrackingViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 NextText	Returns or sets a string of text that is displayed in the "Next Page" link when the grid allows paging.
 PageCount	Returns or sets a value that specifies the total number of data pages that exist when paging is enabled. Setting of the value makes sense only if you use custom paging. In the regular mode the property is set automatically.
 PagerAppearance	Returns or sets a value that specifies where the Pager class information is rendered to the client.
 PageSize	Returns or sets a value that specifies the number of records per data page that are displayed when paging is enabled.
 Pattern	The pager pattern string.
 PrevText	Returns or sets a string of text that is displayed in the "Previous Page" link when the grid allows paging.
 QuickPages	Number of quick pages.
 Style	Returns a reference to or sets a GridItemStyle object that determines how the Pager links are rendered on the client.
 StyleMode	Returns or sets a value that specifies the type of Pager user interface to render on the client.

Public Methods

 Clone	Creates a new Pager object with all of the property settings of the current object. The new object is functionally identical to the current one.
 CopyFrom	Duplicates the properties of the specified Pager into the instance of the Pager class from which this method is invoked.
 CopyFromSerializable	Copies the property settings of the passed-in Pager object to the current Pager object. Any existing settings in the current object are replaced. This method is serializable.
 Reset	Overridden. Resets all properties of the Pager class to their default values.
 ToString	Overridden. Returns a string representation of a Pager object.
 TrackViewState	Overridden.

Protected Properties

 ViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
--	--

See Also

[Pager Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

RendererConsumerBase Class

Base class implementing [IPlugInConsumer](#) that supports pluggable external renderers for UltraWebGrid.

For a list of all members of this type, see [RendererConsumerBase members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.RendererConsumerBase

Syntax

```
[Visual Basic]
Public Class RendererConsumerBase
    Implements IPlugInConsumer
```

```
[C#]
public class RendererConsumerBase : IPlugInConsumer
```

```
[JScript]
public class RendererConsumerBase implements IPlugInConsumer
```


See Also

[RendererConsumerBase Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)


RendererConsumerBase Class Members

[RendererConsumerBase overview](#)

Public Constructors

 RendererConsumerBase Constructor	Creates a basic plug-in consumer for external renderers.
---	--

Public Properties

 Location	Specifies the current location of HTML rendering for this server control.
---	---

Public Methods

 RenderExternal	Renders content generated externally from UltraWebGrid into the HTML at the current Location.
---	---

See Also

[RendererConsumerBase Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

RowEventArgs Class

Event arguments that are passed to event handlers that involve changes to rows.

For a list of all members of this type, see [RowEventArgs members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)
[System.EventArgs](#)
[Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)
Infragistics.WebUI.UltraWebGrid.RowEventArgs

Syntax

```
[Visual Basic]  
Public Class RowEventArgs  
    Inherits UltraGridEventArgs
```

```
[C#]  
public class RowEventArgs : UltraGridEventArgs
```

```
[JScript]  
public class RowEventArgs extends UltraGridEventArgs
```




See Also

[RowEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

RowEventArgs Class Members

[RowEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 Data	Returns the Data object associated with the event.
 Row	Returns the Row object associated with the event.

See Also

[RowEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

RowsCollection Class

The Rows Collection manages a set of UltraGridRow objects which are the child rows of an individual Row. For a list of all members of this type, see [RowsCollection members](#).

Object Model

RowsCollection — **UltraGridRow**

Inheritance Hierarchy

[System.Object](#)

[System.Collections.CollectionBase](#)

[Infragistics.WebUI.UltraWebGrid.KeyedObjectCollectionBase](#)

Infragistics.WebUI.UltraWebGrid.RowsCollection

Syntax

```
[Visual Basic]
Public Class RowsCollection
    Inherits KeyedObjectCollectionBase
    Implements IList, ICollection, IEnumerable, IStateManager
```

```
[C#]
public class RowsCollection : KeyedObjectCollectionBase,
    IList, ICollection, IEnumerable, IStateManager
```

```
[JScript]
public class RowsCollection extends KeyedObjectCollectionBase implements
    IList, ICollection, IEnumerable, IStateManager
```

Remarks

The UltraGridRow objects in a Rows Collection make up a *data island*. The top-level Rows of a grid are all part of a single data island.

If the grid is hierarchical, then one or more top-level UltraGridRow objects will have its own Rows Collection, which contains one or more Row objects. This creates a tree structure of parent rows and child rows that is used to display hierarchical data.


See Also

[RowsCollection Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)







RowsCollection Class Members

[RowsCollection overview](#)
















Public Constructors




 RowsCollection Constructor	Overloaded.
---	-------------

Public Properties



 All (Inherited from KeyedObjectCollectionBase)	The collection as an array of objects
 Band	Provides access to the UltraGridBand object that contains this collection of rows.
 Count (Inherited from System.Collections.CollectionBase)	
 IsReadOnly	Overridden. Returns a Boolean value indicating whether the collection is read-only.
 Item	Property indexer for the Rows Collection.
 Owner (Inherited from KeyedObjectCollectionBase)	Provides public access to the owning object of this collection

Public Methods

 Add	Overloaded. Creates and adds an UltraGridRow object to the collection.
 Clear (Inherited from System.Collections.CollectionBase)	Removes all the UltraGridRow objects from the collection.
 Contains (Inherited from KeyedObjectCollectionBase)	Returns true if the collection contains this item
 CopyTo (Inherited from KeyedObjectCollectionBase)	Copies the items into the array
 Exists (Inherited from KeyedObjectCollectionBase)	Returns true if an object with this key is already in the collection. Note, if key is null or a zero length string this method returns false
 FromKey	Key string indexer for the collection. Uses key string to return a row.
 GetEnumerator	Returns an enumerator object for the Rows collection.
 GetFooterText	Overloaded.
 GetItem (Inherited from KeyedObjectCollectionBase)	Overloaded.
 GetNextRowDepthFirst	This method returns the next UltraGridRow within the grid's overall hierarchical structure.
 GetPreviousRowDepthFirst	This method returns the previous UltraGridRow within the grid's overall hierarchical structure.
 IndexOf (Inherited from KeyedObjectCollectionBase)	Overloaded.
 Insert	Overloaded.
 Remove	Removes the specified UltraGridRow object from the collection.
 RemoveAt (Inherited from System.Collections.CollectionBase)	Removes the UltraGridRow object at the specified index from the collection.

 SetFooterText	Overloaded.
 Sort	Sorts the Rows collection.
 ValidateKeyDoesNotExist (Inherited from KeyedObjectCollectionBase)	Overloaded.

Protected Properties

 InnerList (Inherited from System.Collections.CollectionBase)	
 List (Inherited from System.Collections.CollectionBase)	

Protected Methods

 CreateArray (Inherited from KeyedObjectCollectionBase)	Virtual method used by the All 'get' method to create the array it returns.
 InternalAdd	Overloaded.
 InternalClear (Inherited from KeyedObjectCollectionBase)	Clears the collection
 InternalInsert (Inherited from KeyedObjectCollectionBase)	Inserts an object into the collection
 InternalRemove (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 InternalRemoveAt (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 OnClear (Inherited from System.Collections.CollectionBase)	
 OnClearComplete (Inherited from System.Collections.CollectionBase)	
 OnInsert (Inherited from System.Collections.CollectionBase)	
 OnInsertComplete (Inherited from System.Collections.CollectionBase)	
 OnRemove (Inherited from System.Collections.CollectionBase)	
 OnRemoveComplete (Inherited from System.Collections.CollectionBase)	
 OnSet (Inherited from System.Collections.CollectionBase)	
 OnSetComplete (Inherited from System.Collections.CollectionBase)	
 OnValidate (Inherited from System.Collections.CollectionBase)	

See Also

[RowsCollection Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

RowsEditor Class

Summary description for ColumnsEditor.

For a list of all members of this type, see [RowsEditor members](#).

Inheritance Hierarchy

[System.Object](#)

[System.Drawing.Design.UITypeEditor](#)

Infragistics.WebUI.UltraWebGrid.RowsEditor

Syntax

```
[Visual Basic]
```

```
Public Class RowsEditor  
    Inherits UITypeEditor
```

```
[C#]
```

```
public class RowsEditor : UITypeEditor
```

```
[JScript]
```

```
public class RowsEditor extends UITypeEditor
```

See Also

[RowsEditor Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)





RowsEditor Class Members

[RowsEditor overview](#)

Public Constructors

 RowsEditor Constructor	
---	--

Public Methods

 EditValue	Overloaded. Overridden.
 GetEditStyle	Overloaded. Overridden.
 GetPaintValueSupported	Overloaded. Overridden.
 PaintValue (Inherited from System.Drawing.Design.UITypeEditor)	Overloaded.

See Also

[RowsEditor Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SelectedCellsCollection Class

This collection manages the UltraGridCell objects that have been selected with the mouse on the client. For a list of all members of this type, see [SelectedCellsCollection members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

[System.Collections.CollectionBase](#)

Infragistics.WebUI.UltraWebGrid.SelectedCellsCollection

Syntax

```
[Visual Basic]
Public Class SelectedCellsCollection
    Inherits CollectionBase
    Implements IList, ICollection, IEnumerable, IStateManager
```

```
[C#]
public class SelectedCellsCollection : CollectionBase,
IList, ICollection, IEnumerable, IStateManager
```

```
[JScript]
public class SelectedCellsCollection extends CollectionBase implements
IList, ICollection, IEnumerable, IStateManager
```


See Also

[SelectedCellsCollection Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)




SelectedCellsCollection Class Members

[SelectedCellsCollection overview](#)










Public Constructors

 SelectedCellsCollection Constructor	Overloaded.
--	-------------



Public Properties

 Count (Inherited from System.Collections.CollectionBase)	
 IsReadOnly	Returns a Boolean value indicating whether the collection is read-only.
 Item	Property indexer for the SelectedRowsCollection.





Public Methods






 Add	Inserts an UltraGridCell object into the collection.
 Clear (Inherited from System.Collections.CollectionBase)	Removes all UltraGridCell objects from the collection.
 Contains	Determines whether the collection contains the specified UltraGridCell object.
 CopyTo	
 GetEnumerator	Returns an enumerator object for the SelectedCells collection.
 IndexOf	
 Insert	Inserts an UltraGridCell object into the collection at the specified index.
 Remove	Removes the specified UltraGridCell object from the collection.
 RemoveAt (Inherited from System.Collections.CollectionBase)	Removes the UltraGridCell object at the specified index from the collection.

Protected Properties

 InnerList (Inherited from System.Collections.CollectionBase)	
 List (Inherited from System.Collections.CollectionBase)	

Protected Methods

 OnClear (Inherited from System.Collections.CollectionBase)	
 OnClearComplete (Inherited from System.Collections.CollectionBase)	
 OnInsert (Inherited from System.Collections.CollectionBase)	
 OnInsertComplete (Inherited from System.Collections.CollectionBase)	

 OnRemove (Inherited from System.Collections.CollectionBase)	
 OnRemoveComplete (Inherited from System.Collections.CollectionBase)	
 OnSet (Inherited from System.Collections.CollectionBase)	
 OnSetComplete (Inherited from System.Collections.CollectionBase)	
 OnValidate (Inherited from System.Collections.CollectionBase)	

See Also

[SelectedCellsCollection Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SelectedCellsEventArgs Class

Event arguments that are passed to event handlers that involve the selection of cells.

For a list of all members of this type, see [SelectedCellsEventArgs members](#).

Object Model

SelectedCellsEventArgs

SelectedCells (SelectedCellsCollection)

UltraGridColumn

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.UltraGridColumnEventArgs](#)

Infragistics.WebUI.UltraWebGrid.SelectedCellsEventArgs

Syntax

```
[Visual Basic]
Public Class SelectedCellsEventArgs
    Inherits UltraGridColumnEventArgs
```

```
[C#]
public class SelectedCellsEventArgs : UltraGridColumnEventArgs
```

```
[JScript]
public class SelectedCellsEventArgs extends UltraGridColumnEventArgs
```



See Also

[SelectedCellsEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SelectedCellsEventArgs Class Members

[SelectedCellsEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 SelectedCells	

See Also

[SelectedCellsEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SelectedColsCollection Class

This collection manages the UltraGridColumn objects that have been selected with the mouse on the client. For a list of all members of this type, see [SelectedColsCollection members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

[System.Collections.CollectionBase](#)

Infragistics.WebUI.UltraWebGrid.SelectedColsCollection

Syntax

```
[Visual Basic]
Public Class SelectedColsCollection
    Inherits CollectionBase
    Implements IList, ICollection, IEnumerable, IStateManager
```

```
[C#]
public class SelectedColsCollection : CollectionBase,
IList, ICollection, IEnumerable, IStateManager
```

```
[JScript]
public class SelectedColsCollection extends CollectionBase implements
IList, ICollection, IEnumerable, IStateManager
```


See Also

[SelectedColsCollection Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)




SelectedColsCollection Class Members

[SelectedColsCollection overview](#)










Public Constructors

 SelectedColsCollection Constructor	Overloaded.
---	-------------



Public Properties

 Count (Inherited from System.Collections.CollectionBase)	
 IsReadOnly	Returns a Boolean value indicating whether the collection is read-only.
 Item	Property indexer for the SelectedCols Collection.





Public Methods






 Add	Inserts an UltraGridColumn object into the collection.
 Clear (Inherited from System.Collections.CollectionBase)	Removes all UltraGridColumn objects from the collection.
 Contains	Determines whether the collection contains the specified UltraGridColumn object.
 CopyTo	
 GetEnumerator	Returns an enumerator object for the collection.
 IndexOf	
 Insert	Inserts an UltraGridColumn object into the collection at the specified index.
 Remove	Removes the specified UltraGridColumn object from the collection.
 RemoveAt (Inherited from System.Collections.CollectionBase)	Removes the UltraGridColumn object at the specified index from the collection.

Protected Properties

 InnerList (Inherited from System.Collections.CollectionBase)	
 List (Inherited from System.Collections.CollectionBase)	

Protected Methods

 OnClear (Inherited from System.Collections.CollectionBase)	
 OnClearComplete (Inherited from System.Collections.CollectionBase)	
 OnInsert (Inherited from System.Collections.CollectionBase)	
 OnInsertComplete (Inherited from System.Collections.CollectionBase)	

 OnRemove (Inherited from System.Collections.CollectionBase)	
 OnRemoveComplete (Inherited from System.Collections.CollectionBase)	
 OnSet (Inherited from System.Collections.CollectionBase)	
 OnSetComplete (Inherited from System.Collections.CollectionBase)	
 OnValidate (Inherited from System.Collections.CollectionBase)	

See Also

[SelectedColsCollection Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SelectedColumnsEventArgs Class

Event arguments that are passed to event handlers that involve the selection of columns.

For a list of all members of this type, see [SelectedColumnsEventArgs members](#).

Object Model

SelectedColumnsEventArgs

SelectedColumns (SelectedColsCollection)

UltraGridColumn

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)

Infragistics.WebUI.UltraWebGrid.SelectedColumnsEventArgs

Syntax

```
[Visual Basic]
Public Class SelectedColumnsEventArgs
    Inherits UltraGridEventArgs
```

```
[C#]
public class SelectedColumnsEventArgs : UltraGridEventArgs
```

```
[JScript]
public class SelectedColumnsEventArgs extends UltraGridEventArgs
```



See Also

[SelectedColumnsEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SelectedColumnsEventArgs Class Members

[SelectedColumnsEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 SelectedColumns	

See Also

[SelectedColumnsEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SelectedRowsCollection Class

This collection manages the UltraGridRow objects that have been selected with the mouse on the client. For a list of all members of this type, see [SelectedRowsCollection members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)
[System.Collections.CollectionBase](#)
Infragistics.WebUI.UltraWebGrid.SelectedRowsCollection

Syntax

```
[Visual Basic]  
Public Class SelectedRowsCollection  
    Inherits CollectionBase  
    Implements IList, ICollection, IEnumerable, IStateManager
```

```
[C#]  
public class SelectedRowsCollection : CollectionBase,  
IList, ICollection, IEnumerable, IStateManager
```

```
[JScript]  
public class SelectedRowsCollection extends CollectionBase implements  
IList, ICollection, IEnumerable, IStateManager
```


See Also

[SelectedRowsCollection Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)




SelectedRowsCollection Class Members

[SelectedRowsCollection overview](#)










Public Constructors

 SelectedRowsCollection Constructor	Overloaded.
---	-------------



Public Properties

 Count (Inherited from System.Collections.CollectionBase)	
 IsReadOnly	Returns a Boolean value indicating whether the collection is read-only.
 Item	Property indexer for the SelectedRows collection.





Public Methods






 Add	Inserts an UltraGridRow object into the collection.
 Clear (Inherited from System.Collections.CollectionBase)	Removes all UltraGridRow objects from the collection.
 Contains	Determines whether the collection contains the specified UltraGridRow object.
 CopyTo	
 GetEnumerator	Returns an enumerator object for the collection.
 IndexOf	Returns index of the specified UltraGridRow object or -1 if the collection does not contain the row.
 Insert	Inserts an UltraGridRow object into the collection at the specified index.
 Remove	Removes the specified UltraGridRow object from the collection.
 RemoveAt (Inherited from System.Collections.CollectionBase)	Removes the UltraGridRow object at the specified index from the collection.

Protected Properties

 InnerList (Inherited from System.Collections.CollectionBase)	
 List (Inherited from System.Collections.CollectionBase)	

Protected Methods

 OnClear (Inherited from System.Collections.CollectionBase)	
 OnClearComplete (Inherited from System.Collections.CollectionBase)	
 OnInsert (Inherited from System.Collections.CollectionBase)	
 OnInsertComplete (Inherited from System.Collections.CollectionBase)	

 OnRemove (Inherited from System.Collections.CollectionBase)	
 OnRemoveComplete (Inherited from System.Collections.CollectionBase)	
 OnSet (Inherited from System.Collections.CollectionBase)	
 OnSetComplete (Inherited from System.Collections.CollectionBase)	
 OnValidate (Inherited from System.Collections.CollectionBase)	

See Also

[SelectedRowsCollection Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SelectedRowsEventArgs Class

Event arguments that are passed to event handlers that involve the selection of rows.

For a list of all members of this type, see [SelectedRowsEventArgs members](#).

Object Model

SelectedRowsEventArgs

SelectedRows (SelectedRowsCollection)

UltraGridRow

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)

Infragistics.WebUI.UltraWebGrid.SelectedRowsEventArgs

Syntax

```
[Visual Basic]
Public Class SelectedRowsEventArgs
    Inherits UltraGridEventArgs
```

```
[C#]
public class SelectedRowsEventArgs : UltraGridEventArgs
```

```
[JScript]
public class SelectedRowsEventArgs extends UltraGridEventArgs
```



See Also

[SelectedRowsEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SelectedRowsEventArgs Class Members

[SelectedRowsEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 SelectedRows	

See Also

[SelectedRowsEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SortColumnEventArgs Class

Event arguments that are passed to event handlers that involve column sorting.

For a list of all members of this type, see [SortColumnEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)

Infragistics.WebUI.UltraWebGrid.SortColumnEventArgs

Syntax

```
[Visual Basic]
Public Class SortColumnEventArgs
    Inherits UltraGridEventArgs
```

```
[C#]
public class SortColumnEventArgs : UltraGridEventArgs
```

```
[JScript]
public class SortColumnEventArgs extends UltraGridEventArgs
```





See Also

[SortColumnEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SortColumnEventArgs Class Members

[SortColumnEventArgs overview](#)

Public Properties

 BandNo	The index of the Band in which the sorting is taking place. -1 if multiple sorting is done on the client.
 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 ColumnNo	The index of the Column in which the sorting is taking place. -1 if multiple sorting is done on the client.
 ShiftKey	Shows if the shift key is pressed while the header is clicked.

See Also

[SortColumnEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SortedColsCollection Class

The collection of UltraGridColumn objects that are being used to sort the data in the band.
For a list of all members of this type, see [SortedColsCollection members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

[System.Collections.CollectionBase](#)

[Infragistics.WebUI.UltraWebGrid.KeyedObjectCollectionBase](#)

[Infragistics.WebUI.UltraWebGrid.ColumnsCollection](#)

Infragistics.WebUI.UltraWebGrid.SortedColsCollection

Syntax

```
[Visual Basic]
Public Class SortedColsCollection
    Inherits ColumnsCollection
    Implements IList, ICollection, IEnumerable, IStateManager, ISupportRollback
```

```
[C#]
public class SortedColsCollection : ColumnsCollection,
IList, ICollection, IEnumerable, IStateManager, ISupportRollback
```

```
[JScript]
public class SortedColsCollection extends ColumnsCollection implements
IList, ICollection, IEnumerable, IStateManager, ISupportRollback
```

Remarks

You can sort data in a Band on a column-by-column basis by choosing the columns you want to use as sort criteria and sorting them. As you choose a column to sort on, its UltraGridColumn object is added to the **SortedCols** collection. You can use this collection to determine which columns that are being used to sort the Band and to access the UltraGridColumn objects for those columns as needed.

See Also

[SortedColsCollection Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)








SortedColsCollection Class Members

[SortedColsCollection overview](#)











Public Constructors




 SortedColsCollection Constructor	
---	--

Public Properties



 All (Inherited from KeyedObjectCollectionBase)	The collection as an array of objects
 Band (Inherited from ColumnsCollection)	Provides access to the UltraGridBand object that contains this Columns collection
 Count (Inherited from System.Collections.CollectionBase)	
 HasChanges (Inherited from ColumnsCollection)	
 IsReadOnly (Inherited from ColumnsCollection)	Returns a Boolean value indicating whether the collection is read-only.
 Item (Inherited from ColumnsCollection)	Property indexer for the collection. Uses numeric index.
 Owner (Inherited from KeyedObjectCollectionBase)	Provides public access to the owning object of this collection

Public Methods















 Add	Overloaded. Overridden.
 Clear	Overloaded. Overridden. Removes all UltraGridColumn objects from the collection.
 Commit (Inherited from ColumnsCollection)	
 Contains (Inherited from KeyedObjectCollectionBase)	Returns true if the collection contains this item
 CopyFrom (Inherited from ColumnsCollection)	Populates a Columns collection with all of the UltraGridColumn objects in a different Columns collection.
 CopyTo (Inherited from KeyedObjectCollectionBase)	Copies the items into the array
 CreateBackup (Inherited from ColumnsCollection)	
 Exists (Inherited from KeyedObjectCollectionBase)	Returns true if an object with this key is already in the collection. Note, if key is null or a zero length string this method returns false
 FromKey (Inherited from ColumnsCollection)	Key string indexer for the collection. Uses key string to return a column.
 GetEnumerator (Inherited from ColumnsCollection)	Returns an enumerator object for the Columns collection.
 GetItem (Inherited from KeyedObjectCollectionBase)	Overloaded.
 IndexOf (Inherited from KeyedObjectCollectionBase)	Overloaded.
 Insert	Overloaded. Overridden.
 Remove	Overloaded. Overridden.

 RemoveAt	Overloaded. Overridden.
 Rollback (Inherited from ColumnsCollection)	
 ValidateKeyDoesNotExist (Inherited from KeyedObjectCollectionBase)	Overloaded.

Protected Properties

 InnerList (Inherited from System.Collections.CollectionBase)	
 List (Inherited from System.Collections.CollectionBase)	

Protected Methods

 CreateArray (Inherited from KeyedObjectCollectionBase)	Virtual method used by the All 'get' method to create the array it returns.
 InternalAdd (Inherited from KeyedObjectCollectionBase)	Appends the object to the collection
 InternalClear (Inherited from KeyedObjectCollectionBase)	Clears the collection
 InternalInsert (Inherited from KeyedObjectCollectionBase)	Inserts an object into the collection
 InternalRemove (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 InternalRemoveAt (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 OnClear (Inherited from System.Collections.CollectionBase)	
 OnClearComplete (Inherited from System.Collections.CollectionBase)	
 OnInsert (Inherited from System.Collections.CollectionBase)	
 OnInsertComplete (Inherited from System.Collections.CollectionBase)	
 OnRemove (Inherited from System.Collections.CollectionBase)	
 OnRemoveComplete (Inherited from System.Collections.CollectionBase)	
 OnSet (Inherited from System.Collections.CollectionBase)	
 OnSetComplete (Inherited from System.Collections.CollectionBase)	
 OnValidate (Inherited from System.Collections.CollectionBase)	

See Also

[SortedColsCollection Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

SpecialBoxBase Class

Base class for all box classes. It contains all common properties/methods

For a list of all members of this type, see [SpecialBoxBase members](#).

Object Model

SpecialBoxBase

Style (GridItemStyle)

Inheritance Hierarchy

[System.Object](#)

[Infragistics.WebUI.Shared.WebComponentBase](#)

Infragistics.WebUI.UltraWebGrid.SpecialBoxBase

[Infragistics.WebUI.UltraWebGrid.GroupByBox](#)

[Infragistics.WebUI.UltraWebGrid.AddNewBox](#)

Syntax

```
[Visual Basic]
Public Class SpecialBoxBase
    Inherits WebComponentBase
    Implements IStateManager, ISupportRollback
```

```
[C#]
public class SpecialBoxBase : WebComponentBase, IStateManager, ISupportRollback
```

```
[JScript]
public class SpecialBoxBase extends WebComponentBase implements
IStateManager, ISupportRollback
```

See Also

[SpecialBoxBase Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)







SpecialBoxBase Class Members

[SpecialBoxBase overview](#)








Public Constructors

 SpecialBoxBase Constructor	
---	--


Public Properties

 ButtonConnectorColor	Returns a reference to or sets a Color object that specifies the color of the button connector lines for an object.
 ButtonConnectorStyle	Returns a reference to or sets a BorderStyle object that specifies the style of the button connector lines for an object.
 HasChanges (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 HasStyle	Returns a Boolean value that determines whether the Style property is currently set to a GridItemStyle object.
 IsTrackingViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 Style	Returns a reference to or sets a GridItemStyle object that determines how a button object is rendered on the client.

Public Methods

 Commit (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 CopyFrom	
 CreateBackup (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 Reset	Overridden. Resets all properties of the SpecialBoxBase class to their default values.
 Rollback (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 ToString (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
 TrackViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	

Protected Properties

 ViewState (Inherited from Infragistics.WebUI.Shared.WebComponentBase)	
--	--

See Also

[SpecialBoxBase Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

Strings Class

This class provides a common location for string that are used in the grid.

For a list of all members of this type, see [Strings members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.Strings

Syntax

```
[Visual Basic]
Public Class Strings
    Implements IStateManager
```

```
[C#]
public class Strings : IStateManager
```

```
[JScript]
public class Strings implements IStateManager
```

See Also

[Strings Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)


Strings Class Members

[Strings overview](#)





Public Constructors

 Strings Constructor	
--	--


Public Properties

 AddNewPrompt	Returns or sets a string of text that is displayed to the left of the AddNewBox button.
 DownLevelCancelPrompt	Returns or sets a string of text that is displayed in the edit column cells when editing is allowed and the grid is in the edit mode in the down level rendering.
 DownLevelDeleteColumnHeader	Returns or sets a string of text that is displayed in the header of the delete column when deletion is allowed in the down level grid.
 DownLevelDeletePrompt	Returns or sets a string of text that is displayed in the delete column cells when deletion is allowed in the down level grid.
 DownLevelEditColumnHeader	Returns or sets a string of text that is displayed in the header of the edit column when editing is allowed in the down level grid.
 DownLevelEditPrompt	Returns or sets a string of text that is displayed in the edit column cells when editing is allowed in the down level grid.
 DownLevelUpdatePrompt	Returns or sets a string of text that is displayed in the edit column cells when editing is allowed and the grid is in the edit mode in the down level rendering.
 GroupByBoxPrompt	Returns or sets a string of text that indicates to the user how to perform GroupBy operations.
 GroupByRowDescriptionMaskDefault	A string that determines what text is shown in GroupBy rows. This is the default value that may be overridden by individual bands.
 NoDataMessage	A string value that specifies the message that should be displayed when the grid contains no data.
 NullTextDefault	Specifies how null values will be represented in the grid. This is the default value that may be overridden by individual band.
 PagerNextText	Returns or sets a string of text that is displayed in the "Next Page" link when the grid allows paging.
 PagerPrevText	Returns or sets a string of text that is displayed in the "Previous Page" link when the grid allows paging.

Public Methods

 Clone	Clone the Strings object
 CopyFrom	Copy the Strings object
 Reset	Resets all properties of the Strings class to their default values.
 ToString	Overridden. Returns a string representation of an Strings object.

Protected Properties

 ViewState	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.
--	--

See Also

[Strings Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

TemplatedColumn Class

The TemplatedColumn class provides access to the template-specific features of the UltraGridColumn object. You can use this class to manage the header, footer and cell templates attached to any given column.

For a list of all members of this type, see [TemplatedColumn members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

[Infragistics.WebUI.UltraWebGrid.KeyedObjectBase](#)

[Infragistics.WebUI.UltraWebGrid.UltraGridColumn](#)

Infragistics.WebUI.UltraWebGrid.TemplatedColumn

Syntax

[Visual Basic]

```
Public Class TemplatedColumn
```

```
    Inherits UltraGridColumn
```

```
    Implements IKeyedObject, IStateManager, INamingContainer
```

[C#]

```
public class TemplatedColumn : UltraGridColumn,
```

```
IKeyedObject, IStateManager, INamingContainer
```

```
[JScript]
public class TemplatedColumn extends UltraGridColumn implements
IKeyedObject, IStateManager, INamingContainer
```

Remarks

Each column can support three kinds of templates. Header templates apply to the column header and determine the text that will be displayed in the header and the image displayed for the sort indicator. Footer templates apply to the column footer and supply text and summary text for the footer. Cell templates are applied to each cell of the column, and offer the capability to provide a custom editing control layout on a cell-by-cell basis.


See Also

[TemplatedColumn Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)















TemplatedColumn Class Members


















[TemplatedColumn overview](#)

Public Constructors

 TemplatedColumn Constructor	Overloaded.
--	-------------





Public Properties











 AllowGroupBy (Inherited from UltraGridColumn)	Determines whether the column can be dragged into the GroupByBox and used to group rows.
 AllowNull (Inherited from UltraGridColumn)	Allows having null values in the column.
 AllowResize (Inherited from UltraGridColumn)	Determines whether a column can be resized and the type of resizing that is permitted.
 AllowResizeResolved (Inherited from UltraGridColumn)	Determines whether a column can be resized and the type of resizing that is permitted. This property will always return the setting that is in control of the column.
 AllowUpdate (Inherited from UltraGridColumn)	Determines whether the values of the column may be updated.
 AllowUpdateResolved (Inherited from UltraGridColumn)	
 Band (Inherited from UltraGridColumn)	Returns the UltraGridBand object of the band that contains the column.
 BaseColumnName (Inherited from UltraGridColumn)	Returns the name of the field in the data store on which the column is based.
 Case (Inherited from UltraGridColumn)	Returns or sets the case to use when editing or displaying column text.
 CellButtonDisplay (Inherited from UltraGridColumn)	Determines how cell buttons will be displayed in the column.
 CellButtonStyle (Inherited from UltraGridColumn)	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the cell buttons of the column.
 CellItems	Returns or sets the collection of CellItems associated with this column. The CellItem functions as the container into which the Cell templates are instantiated.
 CellMultiline (Inherited from UltraGridColumn)	Determines if the cells of the column will edit text as a multiline textarea.
 CellStyle (Inherited from UltraGridColumn)	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the cells of the column.
 CellTemplate	Returns or sets the template that will be used with the column's cells.
 ChangeLinksColor (Inherited from UltraGridColumn)	Allows to apply the cells fore color to the links in the column.
 DataType (Inherited from UltraGridColumn)	The Type of the column.
 DefaultValue (Inherited from UltraGridColumn)	Default value used for cells in this column when a new row is added.
 EditorControlID (Inherited from UltraGridColumn)	A control's ID that is used to edit the cells in the column.

 FieldLen (Inherited from UltraGridColumn)	Maximum field length permitted when editing cells within this column.
 FooterItem	Returns or sets the FooterItem associated with this column. The FooterItem functions as the container into which the cell footer template is instantiated.
 FooterStyle (Inherited from UltraGridColumn)	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the footer section of the column.
 FooterStyleResolved (Inherited from UltraGridColumn)	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the column footer. This object represents the actual visible state of the column footer.
 FooterTemplate	Returns or sets the template that will be used with the column footer.
 FooterText (Inherited from UltraGridColumn)	The text displayed in the footer section of the column.
 FooterTotal (Inherited from UltraGridColumn)	Specifies the type of summary information to display in the column footer. Text, Sum, Average, Minimum, Maximum and Count are available.
 Format (Inherited from UltraGridColumn)	A string used to control the formatting of displayed text.
 HasCellButtonStyle (Inherited from UltraGridColumn)	Determines if a style has been applied to the cell button.
 HasCellStyle (Inherited from UltraGridColumn)	Determines if a style has been applied to the cells of the column.
 HasChanges (Inherited from UltraGridColumn)	
 HasFooterStyle (Inherited from UltraGridColumn)	Determines if a style has been applied to the column footers.
 HasHeaderStyle (Inherited from UltraGridColumn)	Determines if a style has been applied to the column header.
 HasSelectedCellStyle (Inherited from UltraGridColumn)	Determines if a style has been applied to the selected cell.
 HasSelectedHeaderStyle (Inherited from UltraGridColumn)	Determines if a style has been applied to the selected header.
 HasValidators (Inherited from UltraGridColumn)	Determines if the column has a validator list attached.
 HasValueList (Inherited from UltraGridColumn)	Determines if the column has a value list attached.
 HeaderClickAction (Inherited from UltraGridColumn)	Specifies what will occur when the user clicks on a column header.
 HeaderClickActionResolved (Inherited from UltraGridColumn)	Specifies how a column header should react to being clicked. This property will always return the setting that is in control of the column.
 HeaderItem	Returns or sets the HeaderItem associated with this column. The HeaderItem functions as the container into which the column header template is instantiated.
 HeaderStyle (Inherited from UltraGridColumn)	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the header section of the column.
 HeaderTemplate	Returns or sets the template that will be used with the column header.
 HeaderText (Inherited from UltraGridColumn)	The text that appears in the header of the column.
 Hidden (Inherited from UltraGridColumn)	Determines whether a column is hidden from view
 HTMLEncodeContent (Inherited from UltraGridColumn)	



 Index (Inherited from UltraGridColumn)	Index in the Columns collection
 IsBound (Inherited from UltraGridColumn)	Indicates whether this is a bound or an unbound column.
 IsGroupByColumn (Inherited from UltraGridColumn)	True - insert this column into the Band's SortedColumns collection after all existing groupby columns and before any sorted columns that are not Groupbys. The rows that have common values for this column will then be grouped together under a single GroupBy row along with an expansion indicator.
 Key (Inherited from UltraGridColumn)	Unique string identifier for this Column within the Columns collection
 MergeCells (Inherited from UltraGridColumn)	Allows to merge cells with the same value in the column.
 NullText (Inherited from UltraGridColumn)	The string displayed in cells with null values.
 NullTextResolved (Inherited from UltraGridColumn)	The string displayed in cells with null values. This property will always return the setting that is in control of the column.
 Selected (Inherited from UltraGridColumn)	Determines whether the column is selected
 SelectedCellStyle (Inherited from UltraGridColumn)	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the selected cell.
 SelectedHeaderStyle (Inherited from UltraGridColumn)	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the selected column header.
 ServerOnly (Inherited from UltraGridColumn)	Determines whether a column will be rendered down to the client
 SortIndicator (Inherited from UltraGridColumn)	The convenience variable stored for the sorted order of the column. Setting this property will NOT sort the column.
 Tag (Inherited from UltraGridColumn)	A field for storing user-defined, object-related information. Objects assigned to this property must be serializable if ViewState is in effect.
 Type (Inherited from UltraGridColumn)	The type of column that will be rendered.
 Validators (Inherited from UltraGridColumn)	Returns a collection of the validators that are applied to the column.
 ValueList (Inherited from UltraGridColumn)	A list of values that are to be contained in a combo box portion of a column with a combo box style
 Width (Inherited from UltraGridColumn)	The width of the column, in pixels.
 WidthResolved (Inherited from UltraGridColumn)	The width of the column, in pixels. This property will always return the setting that is in control of the column.

Public Methods






 Commit (Inherited from UltraGridColumn)	
 CopyFrom	Overridden. Copies the template from the specified column to the current column. Header, footer and cell templates will be copied.
 CreateBackup (Inherited from UltraGridColumn)	
 Find (Inherited from UltraGridColumn)	Overloaded. Searches for a cell in the column, which value starts with provided string. Case is ignored. Uses previously passed parameters.

 FindNext (Inherited from UltraGridColumn)	Overloaded. Continues search from previously found cell.
 IsEditable (Inherited from UltraGridColumn)	Shows whether cells of the column are editable or not
 IsSelectable (Inherited from UltraGridColumn)	Shows whether the column can be selected or not
 IsSortable (Inherited from UltraGridColumn)	Shows whether the column can be sorted or not
 Move (Inherited from UltraGridColumn)	Moves the column object to the specified position inside the columns collection.
 Reset (Inherited from UltraGridColumn)	Resets all properties of the UltraGridColumn class to their default values.
 ResetKey (Inherited from KeyedObjectBase)	Sets the key back to null
 Rollback (Inherited from UltraGridColumn)	
 ShouldSerializeKey (Inherited from KeyedObjectBase)	Returns true if the key needs to be serialized (not null)
 ToString (Inherited from UltraGridColumn)	Returns a string representation of an UltraGridColumn object.

Protected Properties

 IsTrackingViewState (Inherited from UltraGridColumn)	
 ViewState (Inherited from UltraGridColumn)	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.

Protected Methods

 LoadViewState (Inherited from UltraGridColumn)	
 OnAddedToCollection (Inherited from KeyedObjectBase)	Called when this object is being added to the passed in collection. The default implementation sets the internal primaryCollection reference if it hasn't already been set
 OnRemovedFromCollection (Inherited from KeyedObjectBase)	Called when this object is being removed from the passed in collection. The default implementation nulls out the primaryCollection if the passed in collection matches
 SaveViewState (Inherited from UltraGridColumn)	
 TrackViewState (Inherited from UltraGridColumn)	

See Also

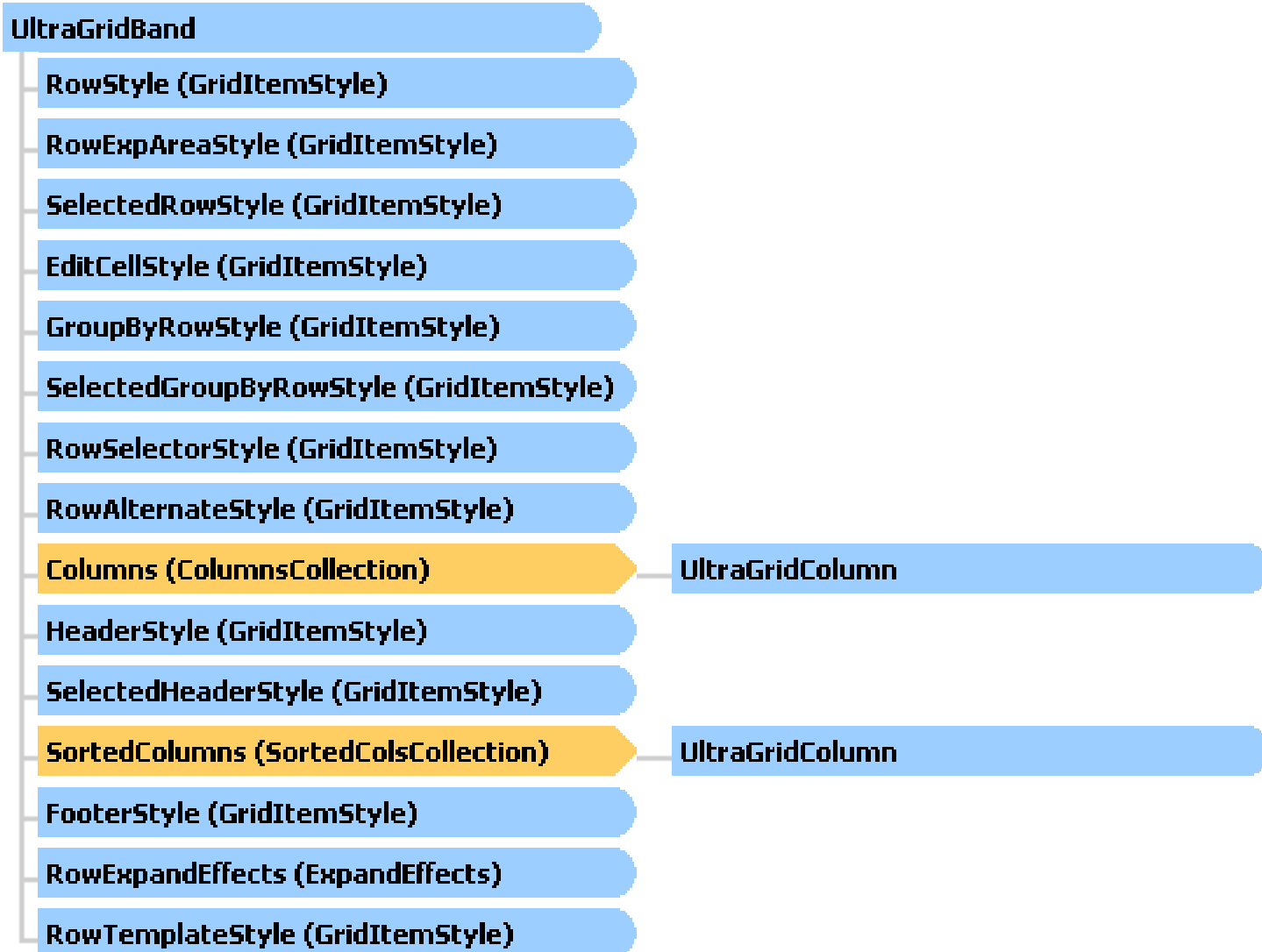
[TemplatedColumn Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraGridBand Class

The UltraGridBand object represents a single level of a hierarchical record set. The columns that make up a band are typically drawn from a single recordsource (table) in the data source.

For a list of all members of this type, see [UltraGridBand members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.KeyedObjectBase

Infragistics.WebUI.UltraWebGrid.UltraGridBand

Syntax

```

[Visual Basic]
Public Class UltraGridBand
    Inherits KeyedObjectBase
    Implements IKeyedObject, IStateManager, ISupportRollback

```

[C#]


```
public class UltraGridBand : KeyedObjectBase, IKeyedObject, IStateManager, ISupportRollback
```

```
[JScript]  
public class UltraGridBand extends KeyedObjectBase implements  
IKeyedObject, IStateManager, ISupportRollback
```

Remarks

The UltraGridBand object represents a set of related columns. These columns generally correspond to the fields in a recordset, but can also include computed columns added by the programmer. A band represents a grouping of all the data at a single level in a hierarchical recordset. In the case of flat data there is only 1 band, whereas a grid with hierarchical data will have multiple bands.

Although the Band object does not directly own any rows within the row hierarchy, it does control the appearance and behavior of those rows within its level of influence.

Example

This example creates the entire grid with the server-side API. All of the bands, columns, rows, and cells are created in code.

[Visual Basic]

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles  
MyBase.Load If IsPostBack Then Exit Sub Dim DataSet As DataSet ' Get  
the data from the database via the middleware With New ReportSystem()  
DataSet = .GetYearlyExpenseReport End With ' Get a pointer to the  
first band Dim Band As Infragistics.WebUI.UltraWebGrid.UltraGridBand = UltraWebGrid1.  
Bands(0) ' Set the band key to Year Band.Key = "Year" ' Set the  
default Column Width and Row Height on the grid Band.DefaultColWidth = New Unit(50)  
Band.DefaultRowHeight = New Unit(20) ' Set the various style properties to  
class in our CSS file Band.HeaderStyle.CssClass = "HeaderClass" Band.  
FooterStyle.CssClass = "FooterClass" Band.RowAlternateStyle.CssClass = "AltRowClass"  
Band.SelectedRowStyle.CssClass = "SelCellClass" Band.RowStyle.CssClass =  
"ItemClass" ' Allow the bands to be expanded by the user Band.Expandable =  
Infragistics.WebUI.UltraWebGrid.Expandable.Yes ' Only show horizontal grid lines in  
our grid Band.GridLines = Infragistics.WebUI.UltraWebGrid.UltraGridLines.Horizontal  
' Set the headers in the band to do a multi-column sort on click Band.  
HeaderClickAction = Infragistics.WebUI.UltraWebGrid.HeaderClickAction.SortMulti ' Do  
not show the row selectors on the end of the grid Band.RowSelectors = Infragistics.  
WebUI.UltraWebGrid.RowSelectors.No Dim Column As System.Data.DataColumn ' Loop  
the columns of the DataSet and build the columns of the grid For Each Column In  
DataSet.Tables("YearlyExpenses").Columns Dim ColumnName As String = Column.  
ColumnName ' Add the Column UltraWebGrid1.Bands.FromKey("Year").  
Columns.Add(ColumnName, ColumnName) ' and set it's size UltraWebGrid1.  
Bands.FromKey("Year").Columns.FromKey(ColumnName).Width = System.Web.UI.WebControls.Unit.Pixel  
(200) If ColumnName <> "Year" Then ' set it to right align  
UltraWebGrid1.Bands.FromKey("Year").Columns.FromKey(ColumnName).CellStyle.  
HorizontalAlign = HorizontalAlign.Right End If Next Dim DataRow As  
DataRow Dim Cell As Infragistics.WebUI.UltraWebGrid.UltraGridCell ' Loop each  
row to get the data to create the cells and rows For Each DataRow In DataSet.Tables  
("YearlyExpenses").Rows() ' Get a new row Dim GridRow As Infragistics.  
WebUI.UltraWebGrid.UltraGridRow = New Infragistics.WebUI.UltraWebGrid.UltraGridRow()  
' Loop each column in the row to get the cell data For Each Column In  
DataRow.Table.Columns ' Create a new cell Cell = New  
Infragistics.WebUI.UltraWebGrid.UltraGridCell() ' If it is not the Year column  
format it as currency If Column.ColumnName <> "Year" Then  
' Put the DataSet column value into the Cell object  
Cell.Value = System.String.Format("{0:C}", DataRow(Column.ColumnName))  
Else ' Put the DataSet column value into the Cell object  
Cell.Value = DataRow(Column.ColumnName).ToString End If  
' Add the cell to the row GridRow.Cells.Add(Cell)
```

```

        Next          ' Add the row to the Grid          UltraWebGrid1.Rows.Add
(GridRow)          Next          UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.
UltraWebGrid.ViewType.Hierarchical          Band = New Infragistics.WebUI.UltraWebGrid.
UltraGridBand()          UltraWebGrid1.Bands.Add(band)          ' Loop the columns of the
DataSet and build the columns of the grid          For Each Column In DataSet.Tables
("YearlyExpenses1").Columns          Dim ColumnName As String = Column.ColumnName
        ' Don't add year to the second band          If ColumnName <> "Year" Then
        ' Add the Column          Band.Columns.Add(ColumnName, ColumnName)
        ' and set it's size          Band.Columns.FromKey(ColumnName).Width =
System.Web.UI.WebControls.Unit.Pixel(200)          If ColumnName <> "Region" Then
        ' set it to right align          Band.Columns.FromKey
(ColumnName).CellStyle.HorizontalAlign = HorizontalAlign.Right          End If
        End If          Next          Dim row As Infragistics.WebUI.UltraWebGrid.
UltraGridRow          For Each row In UltraWebGrid1.Rows          row.Rows.Band = Band
        Next          ' Loop each row to get the data to create the cells and rows          For
Each DataRow In DataSet.Tables("YearlyExpenses1").Rows()          ' Get a new row
        Dim GridRow As Infragistics.WebUI.UltraWebGrid.UltraGridRow = New Infragistics.
WebUI.UltraWebGrid.UltraGridRow()          ' Loop each column in the row to get the cell
data          For Each Column In DataRow.Table.Columns          ' Don't do year
again          If Column.ColumnName <> "Year" Then          ' Create a new
cell          Cell = New Infragistics.WebUI.UltraWebGrid.UltraGridCell()
        ' If it is not the Region column format it as currency
        If Column.ColumnName <> "Region" Then          ' Put the
DataSet column value into the Cell object          Cell.Value = System.String.
Format("{0:C}", DataRow(Column.ColumnName))          Else          '
Put the DataSet column value into the Cell object          Cell.Value = DataRow
(Column.ColumnName).ToString          End If          ' Add the cell to
the row          GridRow.Cells.Add(Cell)          End If          Next
        ' Add the row to the Grid          UltraWebGrid1.Rows(CType(Right(DataRow
("Year").ToString, 1), Int32) - 1).Rows.Add(GridRow)          Next          End Sub

```


See Also

[UltraGridBand Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraGridBand Class Members























[UltraGridBand overview](#)


Public Constructors

















 UltraGridBand Constructor	Overloaded.
--	-------------

Public Properties

 AddButtonCaption	Returns or sets the caption text of the Band's Add button.
 AddButtonToolTipText	Returns or sets the text used as the Add button's tool tip
 AllowAdd	Determines whether the user is allowed to add a new row of data.
 AllowColSizing	Determines whether the user is allowed to change the width of columns.
 AllowColSizingResolved	Determines whether the user is allowed to change the width of columns. This property will always return the setting that is in control of the band.
 AllowColumnMoving	Turns on the ability of moving columns on the server or on the client side.
 AllowColumnMovingResolved	Turns on the ability of moving columns on the server or on the client side. Resolves value considering inheritance from the layout object.
 AllowDelete	Determines whether the user is allowed to delete rows.
 AllowDeleteResolved	Determines whether the user is allowed to delete rows. This property will always return the setting that is in control of the band.
 AllowSorting	Specifies whether the user can sort columns.
 AllowSortingResolved	Specifies whether the user can sort columns. This property will always return the setting that is in control of the band.
 AllowUpdate	Determines whether the user is allowed to update data.
 AllowUpdateResolved	Determines whether the user is allowed to update data. This property will always return the setting that is in control of the band.
 BaseTableName	The name of the table in the data store being used to provide data for the band.
 BorderCollapse	Defines whether the grid borders should collapse or not.
 BorderCollapseResolved	Returns a value that determines whether the grid borders should collapse or not.
 CellClickAction	Specifies what action will occur when a cell is clicked.
 CellPadding	The amount of cell padding used in cells of the band.
 CellPaddingResolved	The amount of cell padding used in cells of the band. This property will always return the setting that is in control of the band.
 CellSpacing	The amount of cell spacing to use in the cells of the band.

 CellSpacingResolved	The amount of cell spacing used in cells of the band. This property will always return the setting that is in control of the band.
 ChildBandColumn	Returns or sets a column name that is to be used in the DataBind
 ColFootersVisible	Determines whether to show column footers.
 ColFootersVisibleResolved	Determines whether to show column footers. This property will always return the setting that is in control of the band.
 ColHeadersVisible	Determines whether to show column headers.
 ColHeadersVisibleResolved	Determines whether to show column headers. This property will always return the setting that is in control of the band.
 CollapseImage	The name of the image used as the "collapse rows" icon.
 Columns	Returns a collection of the columns that make up the band.
 CurrentEditRowImage	The name of the selected editable row image.
 CurrentRowImage	The name of the selected row image.
 DataKeyField	The key field of the table being used as the data source of the band. This property tells UltraWebGrid which column from the database to load into the DataKey property of each Row as it is read and initialized in the grid. Later, the DataKey property of each row can be used as a lookup key to apply changes from the row to the database.
 DefaultColWidth	The default width of columns in the band. This value is overridden by any width settings at the column level.
 DefaultRowHeight	The default height of rows in the band. This value is overridden by any height settings at the row level.
 EditCellStyle	Specifies the style applied to the cell when it is in edit mode.
 Expandable	Determines whether the band can be expanded and collapsed.
 ExpandImage	The name of the image used as the "expand rows" icon.
 FooterStyle	The default style that will be applied to column footers in the band.
 GridID	The control ID of the UltraWebGrid to which the current band belongs.
 GridLines	Specifies which cell borders should be shown in the band.
 GridLinesResolved	Specifies which cell borders should be shown in the band. This property will always return the setting that is in control of the band.
 GroupByRowDescriptionMask	A string that determines what text be shown in GroupBy rows. It can include special substitution strings (e.g. to specify child row count).
 GroupByRowDescriptionMaskResolved	A string that determines what text be shown in GroupBy rows. It can include special substitution strings (e.g. to specify child row count). This property will always return the setting that is in control of the band.

 GroupByRowStyle	An Infragistics.Web.UltraWebGrid.UltraWebStyle object that contains the style properties for the GroupBy rows in the band.
 HasEditCellStyle	Determines if a style has been applied to the edit cell.
 HasFooterStyle	Determines if a style has been applied to the column footers at the band level.
 HasGroupByRowStyle	Determines if a style has been applied to GroupBy rows at the band level.
 HasHeaderStyle	Determines if a style has been applied to the band header.
 HasRowAlternateStyle	
 HasRowExpandEffects	Determines if any of expand effects were applied to the band's row edit template.
 HasRowExpAreaStyle	Determines if a style has been applied to rows expansion area at the band level.
 HasRowSelectorStyle	Determines if a style has been applied to the row selectors at the band level.
 HasRowStyle	Determines if a style has been applied to rows at the band level.
 HasRowTemplateStyle	Determines if a style has been applied to the row edit template at the band level.
 HasSelectedGroupByRowStyle	Determines if a style for selected GroupBy rows has been applied at the band level.
 HasSelectedHeaderStyle	Determines if a style for selected column headers has been applied at the band level.
 HasSelectedRowStyle	Determines if a style for selected rows has been applied at the band level.
 HasSortedColumns	Determines whether the band contains sorted columns.
 HeaderClickAction	Specifies how a column header should react to being clicked.
 HeaderStyle	An Infragistics.Web.UltraWebGrid.UltraWebStyle object that contains the style properties for the band header.
 Hidden	Determines whether the band is visible.
 Indentation	Determines the amount of indenting used for bands.
 IndentationResolved	Determines the amount of indenting used for bands. This property will always return the setting that is in control of the band.
 Index	Returns the index of the band object within the Bands collection
 Key	Overridden. Returns or sets a unique string identifier for this Band within the Bands collection.
 NullText	Specifies how null values will be represented in the grid.
 NullTextResolved	Specifies how null values will be represented in the grid. This property will always return the setting that is in control of the band.
 ParentBand	Returns the UltraGridBand object for the parent of the current band (if it is a child band) or Null for band 0. This property is read-only.

 RowAlternateStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for alternate (even-numbered) rows in the band.
 RowEditItem	Container into which the editing template for a row is instantiated.
 RowEditTemplate	Returns or sets the template that will be used with the rows in the band.
 RowExpandEffects	This object encapsulates the Internet Explorer Transitions functionality that band's row edit template expose.
 RowExpAreaStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the rows expansion area in the band.
 RowSelectors	Returns or sets a value that determines whether row selectors will be displayed.
 RowSelectorsResolved	Returns or sets a value that determines whether row selectors will be displayed. This property will always return the setting that is in control of the band.
 RowSelectorStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the row selectors in the band.
 RowSizing	Specifies whether and how the user can change the height of the rows.
 RowSizingResolved	Specifies whether and how the user can change the height of the rows. This property will always return the setting that is in control of the band.
 RowStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the rows in the band.
 RowTemplateStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the row edit template in the band.
 SelectedGroupByRowStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the GroupBy rows in the band that have been selected.
 SelectedHeaderStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the selected headers in the band.
 SelectedRowStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the rows in the band that are selected.
 SelectTypeCell	Determines the type of selection that may be performed on cells in the band.
 SelectTypeCol	Determines the type of selection that may be performed on columns in the band.
 SelectTypeRow	Determines the type of selection that may be performed on rows in the band.
 SortedColumns	Returns a collection of UltraGridColumn objects that includes all of the columns in the band which have been sorted.



[Tag](#)

A field for storing user-defined, object-related information. If viewstate is enabled, any object assigned to a Tag property must support the ISerializable interface. Basic types are generally serializable but application defined objects may need to implement ISerializable in order to be placed into viewstate.

Public Methods

AddNew	Adds new row to the band.
CopyFrom	Applies the attributes of an existing UltraGridBand object to the current UltraGridBand object, using the property categories specified.
ExpandAll	Overloaded. Expands all rows in the band.
Find	Overloaded. Searches for a cell in the band, which value starts with provided string. Case is ignored. Uses previously passed parameters.
FindNext	Overloaded. Continues search from previously found cell.
GetBatchUpdates	Returns an enumerator that can be used to iterate through all changed rows in the band.
GetRowsEnumerator	Returns an enumerator that can be used to iterate through all rows in the band. The DataChanged property of each row in the enumeration can be tested to see what type of change has occurred (e.g. update, add, delete).
Reset	Resets all properties of the UltraGridBand class to their default values.
ResetKey (Inherited from KeyedObjectBase)	Sets the key back to null
ShouldSerializeKey (Inherited from KeyedObjectBase)	Returns true if the key needs to be serialized (not null)
ToString	Overridden. Returns a string representation of an UltraGridBand object.

Protected Properties

IsTrackingViewState	
ViewState	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.

Protected Methods

OnAddedToCollection (Inherited from KeyedObjectBase)	Called when this object is being added to the passed in collection. The default implementation sets the internal primaryCollection reference if it hasn't already been set
OnRemovedFromCollection (Inherited from KeyedObjectBase)	Called when this object is being removed from the passed in collection. The default implementation nulls out the primaryCollection if the passed in collection matches

See Also

[UltraGridBand Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraGridColumn Class

This class represents a cell in the grid (a specific column in a specific row).

For a list of all members of this type, see [UltraGridColumn members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.KeyedObjectBase

Infragistics.WebUI.UltraWebGrid.UltraGridColumn

Syntax

```

[Visual Basic]
Public Class UltraGridColumn
    Inherits KeyedObjectBase
    Implements IKeyedObject, IStateManager

```

```

[C#]
public class UltraGridColumn : KeyedObjectBase, IKeyedObject, IStateManager

```

```

[JScript]
public class UltraGridColumn extends KeyedObjectBase implements IKeyedObject, IStateManager

```

Example

This example creates the entire grid with the server-side API. All of the bands, columns, rows, and cells are created in code.

[Visual Basic]

```

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    If IsPostBack Then Exit Sub
    Dim DataSet As DataSet ' Get the data from the database via the middleware
    With New ReportSystem()
        DataSet = .GetYearlyExpenseReport End With ' Get a pointer to the first band
        Dim Band As Infragistics.WebUI.UltraWebGrid.UltraGridColumn = UltraWebGrid1.Bands(0) ' Set the band key to Year
        Band.Key = "Year" ' Set the default Column Width and Row Height on the grid
        Band.DefaultColWidth = New Unit(50)
        Band.DefaultRowHeight = New Unit(20) ' Set the various style properties to class in our CSS file
        Band.HeaderStyle.CssClass = "HeaderClass" Band.FooterStyle.CssClass = "FooterClass"
        Band.RowAlternateStyle.CssClass = "AltRowClass"
        Band.SelectedRowStyle.CssClass = "SelCellClass" Band.RowStyle.CssClass =

```



```

"ItemClass"          ' Allow the bands to be expanded by the user          Band.Expandable =
Infragistics.WebUI.UltraWebGrid.Expandable.Yes          ' Only show horizontal grid lines in
our grid          Band.GridLines = Infragistics.WebUI.UltraWebGrid.UltraGridLines.Horizontal
' Set the headers in the band to do a multi-column sort on click          Band.
HeaderClickAction = Infragistics.WebUI.UltraWebGrid.HeaderClickAction.SortMulti          ' Do
not show the row selectors on the end of the grid          Band.RowSelectors = Infragistics.
WebUI.UltraWebGrid.RowSelectors.No          Dim Column As System.Data.DataColumn          ' Loop
the columns of the DataSet and build the columns of the grid          For Each Column In
DataSet.Tables("YearlyExpenses").Columns          Dim ColumnName As String = Column.
ColumnName          ' Add the Column          UltraWebGrid1.Bands.FromKey("Year").
Columns.Add(ColumnName, ColumnName)          ' and set it's size          UltraWebGrid1.
Bands.FromKey("Year").Columns.FromKey(ColumnName).Width = System.Web.UI.WebControls.Unit.Pixel
(200)          If ColumnName <> "Year" Then          ' set it to right align
UltraWebGrid1.Bands.FromKey("Year").Columns.FromKey(ColumnName).CellStyle.
HorizontalAlign = HorizontalAlign.Right          End If          Next          Dim DataRow As
DataRow          Dim Cell As Infragistics.WebUI.UltraWebGrid.UltraGridCell          ' Loop each
row to get the data to create the cells and rows          For Each DataRow In DataSet.Tables
("YearlyExpenses").Rows()          ' Get a new row          Dim GridRow As Infragistics.
WebUI.UltraWebGrid.UltraGridRow = New Infragistics.WebUI.UltraWebGrid.UltraGridRow()
' Loop each column in the row to get the cell data          For Each Column In
DataRow.Table.Columns          ' Create a new cell          Cell = New
Infragistics.WebUI.UltraWebGrid.UltraGridCell()          ' If it is not the Year column
format it as currency          If Column.ColumnName <> "Year" Then
' Put the DataSet column value into the Cell object
Cell.Value = System.String.Format("{0:C}", DataRow(Column.ColumnName))
Else          ' Put the DataSet column value into the Cell object
Cell.Value = DataRow(Column.ColumnName).ToString          End If
' Add the cell to the row          GridRow.Cells.Add(Cell)
Next          ' Add the row to the Grid          UltraWebGrid1.Rows.Add
(GridRow)          Next          UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.
UltraWebGrid.ViewType.Hierarchical          Band = New Infragistics.WebUI.UltraWebGrid.
UltraGridBand()          UltraWebGrid1.Bands.Add(band)          ' Loop the columns of the
DataSet and build the columns of the grid          For Each Column In DataSet.Tables
("YearlyExpenses1").Columns          Dim ColumnName As String = Column.ColumnName
' Don't add year to the second band          If ColumnName <> "Year" Then
' Add the Column          Band.Columns.Add(ColumnName, ColumnName)
' and set it's size          Band.Columns.FromKey(ColumnName).Width =
System.Web.UI.WebControls.Unit.Pixel(200)          If ColumnName <> "Region" Then
' set it to right align          Band.Columns.FromKey
(ColumnName).CellStyle.HorizontalAlign = HorizontalAlign.Right          End If
End If          Next          Dim row As Infragistics.WebUI.UltraWebGrid.
UltraGridRow          For Each row In UltraWebGrid1.Rows          row.Rows.Band = Band
Next          ' Loop each row to get the data to create the cells and rows          For
Each DataRow In DataSet.Tables("YearlyExpenses1").Rows()          ' Get a new row
Dim GridRow As Infragistics.WebUI.UltraWebGrid.UltraGridRow = New Infragistics.
WebUI.UltraWebGrid.UltraGridRow()          ' Loop each column in the row to get the cell
data          For Each Column In DataRow.Table.Columns          ' Don't do year
again          If Column.ColumnName <> "Year" Then          ' Create a new
cell          Cell = New Infragistics.WebUI.UltraWebGrid.UltraGridCell()
' If it is not the Region column format it as currency
If Column.ColumnName <> "Region" Then          ' Put the
DataSet column value into the Cell object          Cell.Value = System.String.
Format("{0:C}", DataRow(Column.ColumnName))          Else          '
Put the DataSet column value into the Cell object          Cell.Value = DataRow
(Column.ColumnName).ToString          End If          ' Add the cell to
the row          GridRow.Cells.Add(Cell)          End If          Next
' Add the row to the Grid          UltraWebGrid1.Rows(CType(Right(DataRow
("Year").ToString, 1), Int32) - 1).Rows.Add(GridRow)          Next          End Sub

```


See Also

[UltraGridCell Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

















UltraGridCell Class Members

[UltraGridCell overview](#)







Public Constructors



 UltraGridCell Constructor	Overloaded.
--	-------------

Public Properties



 Activated	Indicates whether this cell is the active cell.
 AllowEditing	Determines whether the values of the cell may be edited.
 Band	Returns the UltraGridBand object of the band that contains the cell.
 ColSpan	Defines how many columns the cell occupies.
 Column	Returns the UltraGridColumn object of the column that this cell belongs to.
 DataChanged	Indicates whether the cell's data has been edited and not yet committed to the database
 HasStyle	Determines if a style has been applied to the cell.
 Key	Overridden. Unique string identifier for this Cell within the Cells collection
 Row	Returns the UltraGridRow object of the row that contains the cell.
 RowSpan	Defines how many rows the cell occupies.
 Selected	Returns or sets a value that determines whether the cell is selected.
 Style	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the cell.
 Tag	A field for storing user-defined, object-related information. If using ViewState, this object must be serializable.
 Text	Text string associated with the value of a cell.
 Title	Sets tooltip for the cell.
 Value	The value stored in a cell, as it appears in the data source.

Public Methods






 Activate	Sets the current cell as the active cell.
 CopyFrom	Copy the UltraGridCell object
 GetText	Returns the text of the cell taking into account any data masking that has been applied.
 IsEditable	Shows if the cell is editable.
 Reset	Resets all properties of the UltraGridCell class to their default values.
 ResetKey (Inherited from KeyedObjectBase)	Sets the key back to null

 ShouldSerializeKey (Inherited from KeyedObjectBase)	Returns true if the key needs to be serialized (not null)
 ToString	Overridden. Returns a string representation of an UltraGridColumn object.

Protected Properties

 IsTrackingViewState	
 ViewState	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.

Protected Methods

 LoadViewState	
 OnAddedToCollection (Inherited from KeyedObjectBase)	Called when this object is being added to the passed in collection. The default implementation sets the internal primaryCollection reference if it hasn't already been set
 OnRemovedFromCollection (Inherited from KeyedObjectBase)	Called when this object is being removed from the passed in collection. The default implementation nulls out the primaryCollection if the passed in collection matches
 SaveViewState	
 TrackViewState	

See Also

[UltraGridColumn Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraGridColumn Class

The Column object contains the properties and methods that control the look and behavior of a vertical column of cells within UltraWebGrid.

For a list of all members of this type, see [UltraGridColumn members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.KeyedObjectBase

Infragistics.WebUI.UltraWebGrid.UltraGridColumn

[Infragistics.WebUI.UltraWebGrid.TemplatedColumn](#)

Syntax

```
[Visual Basic]
Public Class UltraGridColumn
    Inherits KeyedObjectBase
    Implements IKeyedObject, IStateManager
```

```
[C#]
public class UltraGridColumn : KeyedObjectBase, IKeyedObject, IStateManager
```

```
[JScript]
public class UltraGridColumn extends KeyedObjectBase implements IKeyedObject, IStateManager
```

Remarks

A set of columns in combination with a set of rows comprises a table. When bound to a database, all cells of a column are populated with data from the same column of the database table.

All the cells of a particular column have certain characteristics in common: DataType, Editing format string, Column Selection type, and Column style.

Example

This event is fired when the grid is begin initialized. It gives us the ability to apply and formatting to columns that we want.

[Visual Basic]

```

'-----' Private Sub
UltraWebGrid1_InitializeLayout: ' This event is fired when the grid is begin initialized.
It ' gives us the ability to apply and formatting to columns that ' we want.
'-----' Private Sub
UltraWebGrid1_InitializeLayout(ByVal sender As System.Object, ByVal e As Infragistics.WebUI.
UltraWebGrid.LayoutEventArgs) Handles UltraWebGrid1.InitializeLayout If IsPostBack
Then Exit Sub UltraWebGrid1.DisplayLayout.TableLayout = Infragistics.WebUI.
UltraWebGrid.TableLayout.Fixed UltraWebGrid1.DataKeyField = ExpenseReportData.Tables
(ExpenseReportData.EXPENSE_ITEM_REPORT_TABLE).PrimaryKey(0).ColumnName If Request.
QueryString("ExpenseReportID").ToLower <> "new" Then Dim ErrorColumn As New
Infragistics.WebUI.UltraWebGrid.UltraGridColumn() Dim ErrorMessageColumn As New
Infragistics.WebUI.UltraWebGrid.UltraGridColumn() Dim ImageColumn As New
Infragistics.WebUI.UltraWebGrid.UltraGridColumn() With ErrorColumn
.Key = "Error" .HeaderText = "" .IsBound =
False .AllowUpdate = Infragistics.WebUI.UltraWebGrid.AllowUpdate.No
.Width = New Unit(5) End With With ErrorMessageColumn
.Key = "ErrorMessage" .HeaderText = ""
IsBound = False .AllowUpdate = Infragistics.WebUI.UltraWebGrid.AllowUpdate.No
.Hidden = True End With With ImageColumn
.Width = UltraWebGrid1.Width.Pixel(16) .Key = "Image"
.HeaderText = "" .IsBound = False .AllowUpdate
= Infragistics.WebUI.UltraWebGrid.AllowUpdate.No End With e.Layout.
Bands(0).Columns.Insert(0, ErrorColumn) e.Layout.Bands(0).Columns.Insert(1,
ErrorMessageColumn) e.Layout.Bands(0).Columns.Insert(2, ImageColumn) End
If With e.Layout.Bands(0).Columns.FromKey(ExpenseReportData.PRIMARY_KEY_FIELD)
.Hidden = True End With With e.Layout.Bands(0).Columns.FromKey
(ExpenseReportData.EXPENSE_REPORT_ID_FIELD) .Hidden = True End With
With e.Layout.Bands(0).Columns.FromKey(ExpenseReportData.EXPENSE_ITEM_REPORT_ID_FIELD)
.Hidden = True End With With e.Layout.Bands(0).Columns.FromKey
(ExpenseReportData.RECEIPT_PROVIDED_FIELD) .HeaderText = "Receipt"
Width = UltraWebGrid1.Width.Percentage(8) .CellStyle.HorizontalAlign =
HorizontalAlign.Center End With With e.Layout.Bands(0).Columns.FromKey
(ExpenseReportData.EXPENSE_CATEGORY_FIELD) .Width = UltraWebGrid1.Width.Percentage
(20) .Type = Infragistics.WebUI.UltraWebGrid.ColumnType.DropDownList
.HeaderText = "Expense Category" Dim Category As Infragistics.WebUI.UltraWebGrid.
ValueList .ValueList Category.DataSource = ExpenseReportData
Category.DataMember = ExpenseReportData.EXPENSE_CATEGORY_TABLE
Category.ValueMember = ExpenseReportData.EXPENSE_CATEGORY_FIELD
Category.DisplayMember = ExpenseReportData.EXPENSE_CATEGORY_FIELD
Category.DataBind() End With With e.Layout.Bands(0).Columns.FromKey
(ExpenseReportData.DESCRPTION_FIELD) If Request.QueryString("ExpenseReportID").
ToLower <> "new" Then .Width = UltraWebGrid1.Width.Percentage(35)
Else .Width = UltraWebGrid1.Width.Percentage(50) End
If .CellMultiline = Infragistics.WebUI.UltraWebGrid.CellMultiline.Yes End
With With e.Layout.Bands(0).Columns.FromKey(ExpenseReportData.CREATE_DATE_FIELD)
If Request.QueryString("ExpenseReportID").ToLower <> "new" Then
Width = UltraWebGrid1.Width.Percentage(15) .Format = "MM-dd-yyyy"
.AllowUpdate = Infragistics.WebUI.UltraWebGrid.AllowUpdate.No
.HeaderText = "Create Date" Else .Hidden = True End If
End With With e.Layout.Bands(0).Columns.FromKey(ExpenseReportData.
AMOUNT_FIELD) .Width = UltraWebGrid1.Width.Percentage(20) .Format =

```

```
"$###,###.00" .CellStyle.HorizontalAlign = HorizontalAlign.Right .
Validators.Add(New Infragistics.WebUI.UltraWebGrid.ValidatorItem("AmountValidator"))
.FooterText = "Total:" .FooterTotal = Infragistics.WebUI.UltraWebGrid.
SummaryInfo.Sum .FooterStyle.HorizontalAlign = HorizontalAlign.Right .
FooterStyle.Padding.Left = New Unit(6) End With End Sub
```


See Also

[UltraGridColumn Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)




















UltraGridColumn Class Members





















[UltraGridColumn overview](#)












Public Constructors

 UltraGridColumn Constructor	Overloaded.
--	-------------












Public Properties

 AllowGroupBy	Determines whether the column can be dragged into the GroupByBox and used to group rows.
 AllowNull	Allows having null values in the column.
 AllowResize	Determines whether a column can be resized and the type of resizing that is permitted.
 AllowResizeResolved	Determines whether a column can be resized and the type of resizing that is permitted. This property will always return the setting that is in control of the column.
 AllowUpdate	Determines whether the values of the column may be updated.
 AllowUpdateResolved	
 Band	Returns the UltraGridColumn object of the band that contains the column.
 BaseColumnName	Returns the name of the field in the data store on which the column is based.
 Case	Returns or sets the case to use when editing or displaying column text.
 CellButtonDisplay	Determines how cell buttons will be displayed in the column.
 CellButtonStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the cell buttons of the column.
 CellMultiline	Determines if the cells of the column will edit text as a multiline text area.
 CellStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the cells of the column.
 ChangeLinksColor	Allows to apply the cells foreground color to the links in the column.
 DataType	The Type of the column.
 DefaultValue	Default value used for cells in this column when a new row is added.
 EditorControlID	A control's ID that is used to edit the cells in the column.
 FieldLen	Maximum field length permitted when editing cells within this column.
 FooterStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the footer section of the column.


 FooterStyleResolved	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the column footer. This object represents the actual visible state of the column footer.
 FooterText	The text displayed in the footer section of the column.
 FooterTotal	Specifies the type of summary information to display in the column footer. Text, Sum, Average, Minimum, Maximum and Count are available.
 Format	A string used to control the formatting of displayed text.
 HasCellButtonStyle	Determines if a style has been applied to the cell button.
 HasCellStyle	Determines if a style has been applied to the cells of the column.
 HasFooterStyle	Determines if a style has been applied to the column footers.
 HasHeaderStyle	Determines if a style has been applied to the column header.
 HasSelectedCellStyle	Determines if a style has been applied to the selected cell.
 HasSelectedHeaderStyle	Determines if a style has been applied to the selected header.
 HasValidators	Determines if the column has a validator list attached.
 HasValueList	Determines if the column has a value list attached.
 HeaderClickAction	Specifies what will occur when the user clicks on a column header.
 HeaderClickActionResolved	Specifies how a column header should react to being clicked. This property will always return the setting that is in control of the column.
 HeaderStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the header section of the column.
 HeaderText	The text that appears in the header of the column.
 Hidden	Determines whether a column is hidden from view
 HTMLEncodeContent	
 Index	Index in the Columns collection
 IsBound	Indicates whether this is a bound or an unbound column.
 IsGroupByColumn	True - insert this column into the Band's SortedColumns collection after all existing groupby columns and before any sorted columns that are not Groupbys. The rows that have common values for this column will then be grouped together under a single GroupBy row along with an expansion indicator.
 Key	Overridden. Unique string identifier for this Column within the Columns collection
 MergeCells	Allows to merge cells with the same value in the column.
 NullText	The string displayed in cells with null values.
 NullTextResolved	The string displayed in cells with null values. This property will always return the setting that is in control of the column.

 Selected	Determines whether the column is selected
 SelectedCellStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the selected cell.
 SelectedHeaderStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the selected column header.
 ServerOnly	Determines whether a column will be rendered down to the client
 SortIndicator	The convenience variable stored for the sorted order of the column. Setting this property will NOT sort the column.
 Tag	A field for storing user-defined, object-related information. Objects assigned to this property must be serializable if ViewState is in effect.
 Type	The type of column that will be rendered.
 Validators	Returns a collection of the validators that are applied to the column.
 ValueList	A list of values that are to be contained in a combo box portion of a column with a combo box style
 Width	The width of the column, in pixels.
 WidthResolved	The width of the column, in pixels. This property will always return the setting that is in control of the column.

Public Methods

 CopyFrom	Copy the UltraGridColumn object
 Find	Overloaded. Searches for a cell in the column, which value starts with provided string. Case is ignored. Uses previously passed parameters.
 FindNext	Overloaded. Continues search from previously found cell.
 IsEditable	Shows whether cells of the column are editable or not
 IsSelectable	Shows whether the column can be selected or not
 IsSortable	Shows whether the column can be sorted or not
 Move	Moves the column object to the specified position inside the columns collection.
 Reset	Resets all properties of the UltraGridColumn class to their default values.
 ResetKey (Inherited from KeyedObjectBase)	Sets the key back to null
 ShouldSerializeKey (Inherited from KeyedObjectBase)	Returns true if the key needs to be serialized (not null)
 ToString	Overridden. Returns a string representation of an UltraGridColumn object.






Protected Properties

 IsTrackingViewState	
--	--

 [ViewState](#)

Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.

Protected Methods

 LoadViewState	
 OnAddedToCollection (Inherited from KeyedObjectBase)	Called when this object is being added to the passed in collection. The default implementation sets the internal primaryCollection reference if it hasn't already been set
 OnRemovedFromCollection (Inherited from KeyedObjectBase)	Called when this object is being removed from the passed in collection. The default implementation nulls out the primaryCollection if the passed in collection matches
 SaveViewState	
 TrackViewState	

See Also

[UltraGridColumn Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraGridEventArgs Class

The base class for UltraWebGrid events. It includes a Cancel property which allows the processing of the event to be stopped.

For a list of all members of this type, see [UltraGridEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs

[Infragistics.WebUI.UltraWebGrid.UpdateEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.SortColumnEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.SelectedRowsEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.SelectedColumnsEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.SelectedCellsEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.RowEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.PageEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.LayoutEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.FooterEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ColumnEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ClickEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.CellEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.BandEventArgs](#)

Syntax

```
[Visual Basic]
Public Class UltraGridEventArgs
    Inherits EventArgs
```

```
[C#]
public class UltraGridEventArgs : EventArgs
```

```
[JScript]
public class UltraGridEventArgs extends EventArgs
```

See Also

[UltraGridEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)


UltraGridEventArgs Class Members

[UltraGridEventArgs overview](#)

Public Constructors

 UltraGridEventArgs Constructor	
---	--

Public Properties

 Cancel	Property that allows processing of the event to be canceled.
---	--

See Also

[UltraGridEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraGridLayout Class

For a list of all members of this type, see [UltraGridLayout members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.UltraGridLayout

Syntax

```
[Visual Basic]
Public Class UltraGridLayout
    Implements IStateManager
```

```
[C#]
public class UltraGridLayout : IStateManager
```

```
[JScript]
public class UltraGridLayout implements IStateManager
```


See Also

[UltraGridLayout Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)
















UltraGridLayout Class Members





















[UltraGridLayout overview](#)





















Public Constructors





 UltraGridLayout Constructor	<p>The UltraGridLayout object determines the settings of various properties related to the appearance and behavior of the object. The UltraGridLayout object provides a simple way to maintain multiple layouts for the grid and apply them as needed. You can also save grid layouts and restore them later.</p>
--	--










Public Properties

 ActivationObject	<p>Returns a reference to the ActivationObject object. This property is read-only at design-time and run-time.</p>
 ActiveCell	<p>Returns or sets the active cell. This property is not available at design-time.</p>
 ActiveRow	<p>Returns or sets the active row. This property is not available at design-time.</p>
 AddNewBox	<p>Returns a reference to the AddNewBox object. This property is read-only at design-time and run-time.</p>
 AllowAddNewDefault	<p>Determines whether the user is allowed to add a new row of data. This is the default value for the grid and may be overridden by individual bands.</p>
 AllowColSizingDefault	<p>Determines whether the user is allowed to change the width of columns. This is the default value that may be overridden by individual bands.</p>
 AllowColumnMovingDefault	<p>Enables or disables column moving on either the client-side.</p>
 AllowDeleteDefault	<p>Determines whether the user is allowed to delete rows. This is the default value that may be overridden by individual bands.</p>
 AllowSortingDefault	<p>Specifies whether the user can sort columns. This is the default value that may be overridden by individual bands.</p>
 AllowUpdateDefault	<p>Determines whether the user is allowed to update data. This is the default value that may be overridden by individual bands.</p>
 AutoGenerateColumns	<p>Specifies whether the grid should generate any columns automatically when data binding occurs.</p>
 Bands	<p>A collection of Band objects. When used at the grid level, the collection includes all the Band objects in the control.</p>
 BorderCollapseDefault	<p>Sets or returns a value that specifies whether the grid borders should collapse or not.</p>
 CellClickActionDefault	<p>Specifies what action will occur when a cell is clicked. This is the default value that may be overridden by individual bands.</p>
 CellPaddingDefault	<p>The amount of cell padding to use when rendering the grid. This is the default value that may be overridden by individual bands.</p>







 CellSpacingDefault	The amount of cell spacing to use when rendering the grid. This is the default value that may be overridden by individual bands.
 ClientSideEvents	Object holder for client-side event names.
 ColFootersVisibleDefault	Determines whether to show column footers. This is the default value that may be overridden by individual bands.
 ColHeadersVisibleDefault	Determines whether to show column headers. This is the default value that may be overridden by individual bands.
 ColWidthDefault	The default width of columns in the control, in Units.
 CompactRendering	Specifies how the HTML code for the control will be generated.
 DefaultCentury	Default century that is used for two digit years. Default value is 1900.
 EditCellStyleDefault	Specifies the style applied to the cell when it is in edit mode. This is the default value that may be overridden by individual band.
 EnableInternalRowsManagement	Controls whether all rows are stored in viewstate, or only the rows that apply to the current page.
 ExpandableDefault	Returns or sets a value that determines if bands are expandable by default. This is the default value that may be overridden by individual bands.
 FooterStyleDefault	The default style that will be applied to column footers in the band. This is the default value that may be overridden by individual band.
 FrameStyle	An GridItemStyle object that contains the style properties for the control's frame.
 Grid	Returns a reference to the UltraWebGrid object that contains the Layout.
 GridLinesDefault	Specifies which cell borders should be shown in the band. This is the default value that may be overridden by individual band.
 GroupByBox	Returns a reference to the GroupByBox object. This property is read-only at design-time and run-time.
 GroupByRowDescriptionMaskDefault	A string that determines what text is shown in GroupBy rows. This is the default value that may be overridden by individual bands.
 GroupByRowStyleDefault	The style used to format the GroupBy rows in the band. This is the default value that may be overridden by individual band.
 HasActivationObject	Specifies whether the control has an ActivationObject object.
 HasAddNewBox	Indicates whether the control has an AddNew Box object.
 HasClientSideEvents	Indicates whether client-side events are active for the grid.
 HasEditCellStyleDefault	Determines if a style has been applied to the edit cell. This is the default value that may be overridden by individual band.
 HasFooterStyleDefault	Determines if a style has been applied to the column footers at the layout level. This is the default value that may be overridden by individual band.
 HasFrameStyle	Determines if a style has been applied to the control's frame.

 HasGroupByRowStyleDefault	Determines if a style has been applied to GroupBy rows at the layout level.
 HasHeaderStyleDefault	Determines if a style has been applied to the band header. This is the default value that may be overridden by individual band.
 HasImageUrls	Determines whether images were changed in the grid.
 HasRowAlternateStyleDefault	Determines if a style has been applied to alternate grid rows at the band level. This is the default value that may be overridden by individual band.
 HasRowExpAreaStyleDefault	Determines if a style has been applied to rows. This is the default value that may be overridden by individual band.
 HasRowSelectorStyleDefault	Determines if a style has been applied to the row selectors at the band level. This is the default value that may be overridden by individual band.
 HasRowStyleDefault	Determines if a style has been applied to rows at the band level. This is the default value that may be overridden by individual band.
 HasSelectedGroupByRowStyleDefault	Determines if a style for selected GroupBy rows has been applied at the band level. This is the default value that may be overridden by individual band.
 HasSelectedHeaderStyleDefault	Determines if a style for selected column headers has been applied at the band level. This is the default value that may be overridden by individual band.
 HasSelectedRowStyleDefault	Determines if a style for selected rows has been applied at the band level. This is the default value that may be overridden by individual band.
 HasStrings	Determines whether strings were changed in the grid.
 HeaderClickActionDefault	Specifies how a column header should react to being clicked. This is the default value that may be overridden by individual bands or columns.
 HeaderStyleDefault	The default style that will be applied to headers. This is the default value that may be overridden by individual bands or columns.
 ImageUrls	Object holder for images of the grid.
 IndentationDefault	Determines the amount of indenting used for bands. This is the default value that may be overridden by individual bands.
 JavaScriptFileName	The base name and path for the JavaScript files that are used for client side behavior. The root name of the file, (ig_WebGrid) is used as the base for determining the other files that are contain related functionality. If this root file name is changed, then all other file names must be changed as well.
 JavaScriptFileNameCommon	The relative path to the common javascript file.
 LoadOnDemand	Returns or sets an enumerated value that determines how data is loaded into the grid at runtime.
 MergeStyles	
 Name	The name of the control.


 NoDataMessage	A string value that specifies the message that should be displayed when the grid contains no data. A column structure is considered data, so that if you have columns defined but no rows, then you will see the columns, but if you have no columns and no rows that is considered no data.
 NullTextDefault	Specifies how null values will be represented in the grid. This is the default value that may be overridden by individual band.
 Pager	Returns the Pager object associated with the grid.
 ReadOnly	Determines whether the grid will be in read-only mode.
 RowAlternateStyleDefault	The default style that will be applied to alternate (even numbered) rows. This is the default value that may be overridden by individual band.
 RowExpAreaStyleDefault	The default style that will be applied to rows expansion area. This is the default value that may be overridden by individual band.
 RowHeightDefault	The default height that will be applied to all rows. This is the default value that may be overridden by individual band.
 Rows	Returns a collection of the topmost level of rows in the grid. This collection will either contain all the rows in Band 0 or the top level of GroupBy rows (if GroupBy rows are being used.)
 RowSelectorsDefault	Specifies whether row selectors will be displayed. This is the default value that may be overridden by individual bands.
 RowSelectorStyleDefault	The default style that will be applied to row selectors. This is the default value that may be overridden by individual band.
 RowSizingDefault	Specifies whether and how the user can change the height of the rows. This is the default value that may be overridden by individual bands.
 RowStyleDefault	The default style that will be applied to rows. This is the default value that may be overridden by individual band.
 ScrollBar	Specifies how the scroll bar will be shown.
 ScrollBarView	Customizes the scroll bar view.
 Section508Compliant	Enables accessibility support.
 SelectedCells	Returns a collection of the currently selected cells. Not available at design-time.
 SelectedColumns	Returns a collection of the currently selected columns. Not available at design-time.
 SelectedGroupByRowStyleDefault	The default style that will be applied to GroupBy rows that are selected. This is the default value that may be overridden by individual band.
 SelectedHeaderStyleDefault	The default style that will be applied to selected headers. This is the default value that may be overridden by individual band.
 SelectedRows	Returns a collection of the currently selected rows. Not available at design-time.
 SelectedRowStyleDefault	The default style that will be applied to rows that are selected. This is the default value that may be overridden by individual band.

 SelectTypeCellDefault	Specifies the way cells can be selected on the client. This is the default value that may be overridden by individual bands.
 SelectTypeColDefault	Specifies the way columns can be selected on the client. This is the default value that may be overridden by individual bands.
 SelectTypeRowDefault	Specifies the way rows can be selected on the client. This is the default value that may be overridden by individual bands.
 StationaryMargins	Specifies which parts of the grid will remain stationary when scrolling occurs.
 Strings	Object holder for strings of the grid.
 TabDirection	Defines the way the tab key moves from cell to cell: left to right (default), right to left, top to bottom or bottom to top.
 TableLayout	Specifies the type of table layout in effect.
 Tag	A field for storing user-defined, object-related information. If set, the object must support serialization.
 ViewType	Specifies how the grid will display data.

Public Methods

 Clone	Clones the Layout object, returning a new, identical UltraGridLayout object.
 CopyFrom	Copies the settings of the passed-in object to the current object. Any settings in the current object will be overwritten.
 LoadLayout	Overloaded.
 Reset	Resets all properties of the UltraGridLayout class to their default values.
 SaveLayout	Overloaded. Writes the property settings contained in the UltraGridLayout object to an XML string and returns a reference to the string. The string returned can then be used again at a later time to restore the current appearance and behavior of the grid
 ToString	Overridden. Returns a string representation of an UltraGridLayout object.

Protected Properties

 ViewState	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.
--	--

See Also

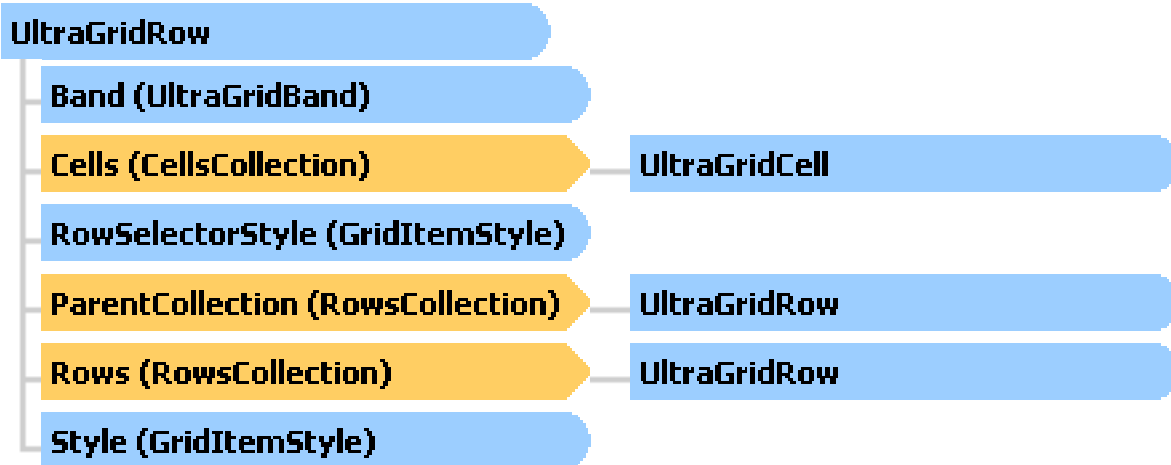
[UltraGridLayout Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraGridRow Class

The UltraGridRow displays a row of data in the grid. A row typically represents the data from a single record in the attached datasource.

For a list of all members of this type, see [UltraGridRow members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.KeyedObjectBase

Infragistics.WebUI.UltraWebGrid.UltraGridRow

[Infragistics.WebUI.UltraWebGrid.GroupByRow](#)

Syntax

```
[Visual Basic]
Public Class UltraGridRow
    Inherits KeyedObjectBase
    Implements IKeyedObject, IComparable, IStateManager

```

```
[C#]
public class UltraGridRow : KeyedObjectBase, IKeyedObject, IComparable, IStateManager
```

```
[JScript]
public class UltraGridRow extends KeyedObjectBase implements
IKeyedObject, IComparable, IStateManager
```

Remarks

The UltraGridRow object represents a single row of data, and corresponds to a single record in the underlying recordset. Rows occupy a position in the data hierarchy of the UltraGrid between Cells and Bands. The UltraGridRow object is always the child of an UltraGridBand object, and its children are UltraGridCell objects.

Much of the data-binding functionality of the grid involves working with the UltraGridRow object. Whenever an UltraGridRow object is loaded by the grid, the **Infragistics.WebUI.UltraWebGrid.InitializeRow** event is fired.

UltraGridRow objects can influence the formatting of the cells they contain through the setting of the UltraGridRow's [Style](#) property. Rows can also be formatted independently of the cells they contain. Frequently,

cells are drawn from the top of the row to the bottom and are aligned edge to edge so that they occupy the entire area of the row; the row itself is not visible because cells are always "above" the row in the grid's z-order. However it is possible to specify spacing between and around cells that lets the underlying UltraGridRow object show through. Only then will formatting applied directly to the UltraGridRow object be visible to the end user.

Example

This example creates the entire grid with the server-side API. All of the bands, columns, rows, and cells are created in code.

[Visual Basic]

```

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    If IsPostBack Then Exit Sub
    Dim DataSet As DataSet
    ' Get
the data from the database via the middleware
    With New ReportSystem()
        DataSet = .GetYearlyExpenseReport
    End With
    ' Get a pointer to the
first band
    Dim Band As Infragistics.WebUI.UltraWebGrid.UltraGridBand = UltraWebGrid1.
Bands(0)
    ' Set the band key to Year
    Band.Key = "Year"
    ' Set the
default Column Width and Row Height on the grid
    Band.DefaultColWidth = New Unit(50)
    Band.DefaultRowHeight = New Unit(20)
    ' Set the various style properties to
class in our CSS file
    Band.HeaderStyle.CssClass = "HeaderClass"
    Band.
FooterStyle.CssClass = "FooterClass"
    Band.RowAlternateStyle.CssClass = "AltRowClass"
    Band.SelectedRowStyle.CssClass = "SelCellClass"
    Band.RowStyle.CssClass =
"ItemClass"
    ' Allow the bands to be expanded by the user
    Band.Expandable =
Infragistics.WebUI.UltraWebGrid.Expandable.Yes
    ' Only show horizontal grid lines in
our grid
    Band.GridLines = Infragistics.WebUI.UltraWebGrid.UltraGridLines.Horizontal
    ' Set the headers in the band to do a multi-column sort on click
    Band.
HeaderClickAction = Infragistics.WebUI.UltraWebGrid.HeaderClickAction.SortMulti
    ' Do
not show the row selectors on the end of the grid
    Band.RowSelectors = Infragistics.
WebUI.UltraWebGrid.RowSelectors.No
    Dim Column As System.Data.DataColumn
    ' Loop
the columns of the DataSet and build the columns of the grid
    For Each Column In
DataSet.Tables("YearlyExpenses").Columns
        Dim ColumnName As String = Column.
ColumnName
        ' Add the Column
        UltraWebGrid1.Bands.FromKey("Year").
Columns.Add(ColumnName, ColumnName)
        ' and set it's size
        UltraWebGrid1.
Bands.FromKey("Year").Columns.FromKey(ColumnName).Width = System.Web.UI.WebControls.Unit.Pixel
(200)
        If ColumnName <> "Year" Then
            ' set it to right align
            UltraWebGrid1.Bands.FromKey("Year").Columns.FromKey(ColumnName).CellStyle.
HorizontalAlign = HorizontalAlign.Right
        End If
    Next
    Dim DataRow As
DataRow
    Dim Cell As Infragistics.WebUI.UltraWebGrid.UltraGridCell
    ' Loop each
row to get the data to create the cells and rows
    For Each DataRow In DataSet.Tables
("YearlyExpenses").Rows()
        ' Get a new row
        Dim GridRow As Infragistics.
WebUI.UltraWebGrid.UltraGridRow = New Infragistics.WebUI.UltraWebGrid.UltraGridRow()
        ' Loop each column in the row to get the cell data
        For Each Column In
DataRow.Table.Columns
            ' Create a new cell
            Cell = New
Infragistics.WebUI.UltraWebGrid.UltraGridCell()
            ' If it is not the Year column
format it as currency
            If Column.ColumnName <> "Year" Then
                ' Put the DataSet column value into the Cell object
                Cell.Value = System.String.Format("{0:C}", DataRow(Column.ColumnName))
            Else
                ' Put the DataSet column value into the Cell object
                Cell.Value = DataRow(Column.ColumnName).ToString
            End If
            ' Add the cell to the row
            GridRow.Cells.Add(Cell)
        Next
        ' Add the row to the Grid
        UltraWebGrid1.Rows.Add
(GridRow)
    Next
    UltraWebGrid1.DisplayLayout.ViewType = Infragistics.WebUI.
UltraWebGrid.ViewType.Hierarchical
    Band = New Infragistics.WebUI.UltraWebGrid.
UltraGridBand()
    UltraWebGrid1.Bands.Add(band)
    ' Loop the columns of the
DataSet and build the columns of the grid
    For Each Column In DataSet.Tables
("YearlyExpenses1").Columns
        Dim ColumnName As String = Column.ColumnName
        ' Don't add year to the second band
        If ColumnName <> "Year" Then
            ' Add the Column
            Band.Columns.Add(ColumnName, ColumnName)
            ' and set it's size
            Band.Columns.FromKey(ColumnName).Width =
System.Web.UI.WebControls.Unit.Pixel(200)
            If ColumnName <> "Region" Then
                ' set it to right align
                Band.Columns.FromKey

```

```

(ColumnName).CellStyle.HorizontalAlign = HorizontalAlign.Right           End If
    End If           Next           Dim row As Infragistics.WebUI.UltraWebGrid.
UltraGridRow           For Each row In UltraWebGrid1.Rows           row.Rows.Band = Band
    Next           ' Loop each row to get the data to create the cells and rows           For
Each DataRow In DataSet.Tables("YearlyExpenses1").Rows()           ' Get a new row
    Dim GridRow As Infragistics.WebUI.UltraWebGrid.UltraGridRow = New Infragistics.
WebUI.UltraWebGrid.UltraGridRow()           ' Loop each column in the row to get the cell
data           For Each Column In DataRow.Table.Columns           ' Don't do year
again           If Column.ColumnName <> "Year" Then           ' Create a new
cell           Cell = New Infragistics.WebUI.UltraWebGrid.UltraGridCell()
    ' If it is not the Region column format it as currency
    If Column.ColumnName <> "Region" Then           ' Put the
DataSet column value into the Cell object           Cell.Value = System.String.
Format("{0:C}", DataRow(Column.ColumnName))           Else           '
Put the DataSet column value into the Cell object           Cell.Value = DataRow
(Column.ColumnName).ToString           End If           ' Add the cell to
the row           GridRow.Cells.Add(Cell)           End If           Next
    ' Add the row to the Grid           UltraWebGrid1.Rows(CType(Right(DataRow
("Year").ToString, 1), Int32) - 1).Rows.Add(GridRow)           Next           End Sub

```


See Also

[UltraGridRow Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)




















UltraGridRow Class Members











[UltraGridRow overview](#)

Public Constructors

















 UltraGridRow Constructor	Overloaded.
---	-------------






Public Properties

 Activated	Indicates whether this row has been activated.
 Band	Returns the UltraGridBand object of the band that contains the row.
 Cells	Returns a collection of the UltraGridCell objects that make up the row.
 DataChanged	Indicates whether the row's data has been edited or this is a new or deleted row and the changes did not yet committed to the database
 DataKey	Returns the key value for the row.
 Expanded	Returns the expanded state of the row.
 HasCells	Determines whether the row contains any cells.
 HasChildRows	Determines whether the row has a set of child rows.
 HasNextSibling	Determines whether a row has a sibling row below it.
 HasParent	Determines whether the row has a parent row or if it is the top-level row.
 HasPrevSibling	Determines whether a row has a sibling row above it.
 HasRowSelectorStyle	Determines if a style has been applied to the row selectors at the row level.
 HasStyle	Determines if a style has been applied to the row at the row level.
 Height	The height of the row, in pixels.
 HeightResolved	The height of the row, in pixels. This property will always return the setting that is in control of the row.
 Hidden	Determines whether the object will be displayed.
 Index	Returns the index of the row object in the UltraGridRows collection
 Key	Overridden. Returns or sets a unique string identifier for this Row within the UltraGridRows collection.
 Level	Returns the number of the band containing the row within the grid's data hierarchy. This property is read-only.
 NextRow	Returns the next sibling row in the band.
 NextVisible	Returns the next visible sibling row in the band.
 ParentCollection	Returns the Rows collection that contains the current row. You can use this collection to enumerate the siblings of the current row.
 ParentRow	Gets the row immediately above the current one in the data hierarchy, if any.


 PrevRow	Returns the previous sibling row in the band.
 PrevVisible	Returns the previous visible sibling row in the band.
 Rows	Returns the collection of the current row's child rows, if any.
 RowSelectorStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the row selector.
 Selected	Determines the selected state of the row.
 ShowExpand	Returns or sets a Boolean value that determines whether a Row object is displayed with an expand image next to it, regardless of whether or not it has children. This property is only effective in conjunction with the use of the LoadOnDemand property of the DisplayLayout object.
 Sizing	Determines whether the row height can be changed.
 SizingResolved	Determines whether the row height can be changed. This property will always return the setting that is in control of the row.
 Style	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the row.
 Tag	A field for storing user-defined, object-related information.

Public Methods






 Activate	Sets the current row as the active row.
 Collapse	Collapses the children of the current row.
 CompareTo	Compares the row to the sorted data in the band and returns the row's position within the sort order.
 CopyFrom	Copy the UltraGridRow object
 Delete	Deletes the current row.
 Expand	Expand the children of the current row.
 ExpandAncestors	Expands all ancestor rows of the current row.
 Find	Overloaded. Searches for a cell in the row, which value starts with provided string. Case is ignored. Uses previously passed parameters.
 FindNext	Overloaded. Continues search from previously found cell.
 GetCellText	Returns string representation of the value of the specified cell.
 GetCellValue	Returns value of the cell in its native form.
 IsActiveRow	Determines whether the row is the active row.
 IsAlternate	Determines whether the row is an alternate (even-numbered) row.
 IsChild	Specifies whether the passed-in row is the direct child of the current row.
 IsDescendant	Specifies whether the passed-in row is a descendant of the current row.
 IsExpanded	Determines whether the row is currently expanded.

 IsSelectable	Determines whether the row may be selected, based on the current selection settings of the grid and the band.
 ResetKey (Inherited from KeyedObjectBase)	Sets the key back to null
 ShouldSerializeKey (Inherited from KeyedObjectBase)	Returns true if the key needs to be serialized (not null)
 Toggle	Obtain the UltraWebGrid object to which the row is attached. Changes the expanded state of the row to Collapsed from Expanded or to Expanded from Collapsed
 ToString	Overridden. Returns a string representation of an UltraGridRow object.

Protected Properties

 ViewState	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.
--	--

Protected Methods

 LoadViewState	
 OnAddedToCollection	Overridden.
 OnRemovedFromCollection (Inherited from KeyedObjectBase)	Called when this object is being removed from the passed in collection. The default implementation nulls out the primaryCollection if the passed in collection matches
 SaveViewState	
 TrackViewState	

See Also

[UltraGridRow Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraGridRowsEnumerator Class

Enumerator for rows in a band.

For a list of all members of this type, see [UltraGridRowsEnumerator members](#).

Object Model

UltraGridRowsEnumerator

Current (UltraGridRow)

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.UltraGridRowsEnumerator

Syntax

```
[Visual Basic]
Public Class UltraGridRowsEnumerator
    Implements IEnumerator
```

```
[C#]
public class UltraGridRowsEnumerator : IEnumerator
```

```
[JScript]
public class UltraGridRowsEnumerator implements IEnumerator
```


See Also

[UltraGridRowsEnumerator Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraGridRowsEnumerator Class Members

[UltraGridRowsEnumerator overview](#)


Public Constructors

 UltraGridRowsEnumerator Constructor	Overloaded.
--	-------------

Public Properties

 Current	
--	--

Public Methods

 MoveNext	
 Reset	

See Also

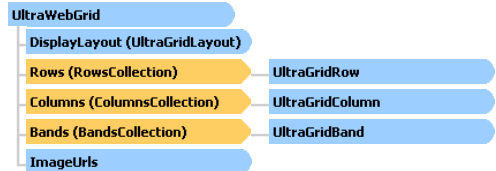
[UltraGridRowsEnumerator Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraWebGrid Class

UltraWebGrid is the top-level object for the control. It provides access to the major functional areas of the control. The DisplayLayout property of UltraWebGrid contains many of the objects used to define behavior and appearance characteristics for the control. The Rows collection property allows access to the top level set of rows of the grid. If there are multiple levels, or Bands, for the grid, each of the top-level rows will allow access to any child rows that may be contained. For basic single level grids, UltraWebGrid exposes the Columns collection and the Rows collection for simple programmable access.

For a list of all members of this type, see [UltraWebGrid members](#).

Object Model



Inheritance Hierarchy

- [System.Object](#)
- [System.Web.UI.Control](#)
- [System.Web.UI.WebControls.WebControl](#)
- [System.Web.UI.WebControls.BaseDataList](#)
- Infragistics.WebUI.UltraWebGrid.UltraWebGrid**

Syntax

```
[Visual Basic]
Public Class UltraWebGrid
    Inherits BaseDataList
    Implements
    IComponent, IDisposable, IParseAccessor, IDataBindingsAccessor, IAttributeAccessor, IPostBackDataHandler, IPostBackEventHandler, IUltraLicensedComponent, INamingContainer, IGetClientSideEvents, IPlugInConsumer, IProvideDesignTimeHtml, ICanEditNavBar, ISupportPresetSerialization, IResolveStyles
```

```
[C#]
public class UltraWebGrid : BaseDataList,
    IComponent, IDisposable, IParseAccessor, IDataBindingsAccessor, IAttributeAccessor, IPostBackDataHandler, IPostBackEventHandler, IUltraLicensedComponent, INamingContainer, IGetClientSideEvents, IPlugInConsumer, IProvideDesignTimeHtml, ICanEditNavBar, ISupportPresetSerialization, IResolveStyles
```

```
[JScript]
public class UltraWebGrid extends BaseDataList implements
    IComponent, IDisposable, IParseAccessor, IDataBindingsAccessor, IAttributeAccessor, IPostBackDataHandler, IPostBackEventHandler, IUltraLicensedComponent, INamingContainer, IGetClientSideEvents, IPlugInConsumer, IProvideDesignTimeHtml, ICanEditNavBar, ISupportPresetSerialization, IResolveStyles
```


See Also

- [UltraWebGrid Members](#)
- [Infragistics.WebUI.UltraWebGrid Namespace](#)

UltraWebGrid Class Members

[UltraWebGrid overview](#)


















Public Constructors

 UltraWebGrid Constructor	Overloaded.
---	-------------

Public Fields

 gridDesigner	
--	--

Public Properties



 Attributes (Inherited from System.Web.UI.WebControls.WebControl)	Gets the collection of arbitrary attributes (for rendering only) that do not correspond to properties on the control.
 BackColor	Overridden. Returns or sets a Color object that specifies the background color of the control.
 Bands	Returns a reference to a BandsCollection collection that contains a collection of UltraGridBand objects.
 BindingContainer (Inherited from System.Web.UI.Control)	
 BorderColor	Overridden. Returns or sets a Color object that specifies the color of the control's borders.
 BorderStyle	Overridden. Returns or sets a BorderStyle object that specifies the style of the control's borders.
 BorderWidth	Overridden. Returns or sets a Unit object that specifies the width of the control's borders.
 Browser	Returns or sets a value that determines how the control is rendered to the client.
 CellPadding	Overridden. Returns or sets a value that specifies the amount of space between cell borders and cell values.
 CellSpacing	Overridden. Returns or sets a value that specifies the amount of space between each cell.
 ClientID (Inherited from System.Web.UI.Control)	Gets the server control identifier generated by ASP.NET.
 Columns	Returns a reference to a ColumnsCollection collection that contains a collection of UltraGridColumn objects.
 Controls	Overridden.
 ControlStyle (Inherited from System.Web.UI.WebControls.WebControl)	Gets the style of the Web server control. This property is used primarily by control developers.
 ControlStyleCreated (Inherited from System.Web.UI.WebControls.WebControl)	Gets a value indicating whether a System.Web.UI.WebControls.Style object has been created for the System.Web.UI.WebControls.WebControl.ControlStyle property. This property is primarily used by control developers.
 DataKeys (Inherited from System.Web.UI.WebControls.BaseDataList)	Gets a System.Web.UI.WebControls.DataKeyCollection that stores the key values of each record (displayed as a row) in a data listing control.
 DataMember (Inherited from System.Web.UI.WebControls.BaseDataList)	Gets or sets the specific data member in a multimember data source to bind to a data listing control.

 DataSource (Inherited from System.Web.UI.WebControls.BaseDataList)	Gets or sets the source containing a list of values used to populate the items within the control.
 DisplayLayout	Returns a reference to or sets the UltraGridLayout object for the grid.
 EnableViewState (Inherited from System.Web.UI.Control)	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client.
 Font	Overridden. Returns a reference to or sets the Font object that specifies the font of the text displayed by the grid.
 ForeColor	Overridden. Returns or sets a Color object that specifies the foreground color of the control.
 GridLines	Overloaded. Returns or sets a value that specifies which cell borders should be displayed for the grid.
 Height	Overridden. Returns or sets a value that specifies the height of the grid.
 HorizontalAlign	Overridden. Returns or sets a value that specifies the horizontal alignment of the text displayed by the grid.
 ID (Inherited from System.Web.UI.Control)	Gets or sets the programmatic identifier assigned to the server control.
 ImageUrls	Object holder for images of the grid.
 IsXmlHttpRequest	Shows that an XML HTTP request is being performed to load more info down to the client.
 JavaScriptFileName	The base name and path for the JavaScript files that are used for client side behavior. The root name of the file, (ig_WebGrid) is used as the base for determining the other files that are contain related functionality. If this root file name is changed, then all other file names must be changed as well.
 JavaScriptFileNameCommon	The relative path to the common javascript file.
 NamingContainer (Inherited from System.Web.UI.Control)	Gets a reference to the server control's naming container, which creates a unique namespace for differentiating between server controls with the same System.Web.UI.Control.ID property value.
 Page (Inherited from System.Web.UI.Control)	Gets a reference to the System.Web.UI.Page instance that contains the server control.
 Parent (Inherited from System.Web.UI.Control)	Gets a reference to the server control's parent control in the page control hierarchy.
 Rows	Returns a reference to a RowsCollection collection that contains a collection of UltraGridRow objects.
 Site (Inherited from System.Web.UI.Control)	Gets information about the Web site to which the server control belongs.
 Style (Inherited from System.Web.UI.WebControls.WebControl)	Gets a collection of text attributes that will be rendered as a style attribute on the outer tag of the Web server control.
 TabIndex (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the tab index of the Web server control.
 TemplateSourceDirectory (Inherited from System.Web.UI.Control)	Gets the virtual directory of the System.Web.UI.Page or System.Web.UI.UserControl that contains the current server control.
 UniqueID (Inherited from System.Web.UI.Control)	Gets the unique, hierarchically-qualified identifier for the server control.
 Visible (Inherited from System.Web.UI.Control)	Gets or sets a value that indicates whether a server control is rendered as UI on the page.

Public Methods














 ApplyPendingInsertion	Overloaded.
 ApplyStyle (Inherited from System.Web.UI.WebControls.WebControl)	Copies any nonblank elements of the specified style to the Web control, overwriting any existing style elements of the control. This method is primarily used by control developers.
 CopyBaseAttributes (Inherited from System.Web.UI.WebControls.WebControl)	Copies the properties not encapsulated by the System.Web.UI.WebControls.WebControl.Style object from the specified Web server control to the Web server control that this method is called from. This method is used primarily by control developers.
 DataBind	Overridden. Binds the grid to the data source specified by the DataSource property.
 Dispose	
 ExpandAll	Overloaded. Expands all rows in the grid.
 FindControl (Inherited from System.Web.UI.Control)	Overloaded.
 GetInvalidCellsEnumerator	Returns InvalidCellsEnumerator which helps to go through all of the cells with invalid value after a post back.
 HasControls (Inherited from System.Web.UI.Control)	Determines if the server control contains any child controls.
 LoadPostData	Called by the framework to load node edits from the client. Returns true to indicate the framework is to call RaisePostDataChangedEvent where the real work gets done.
 LoadPreset	Overloaded.
 MergeStyle	Overloaded.
 OnActiveCellChange	Invoked when the active cell is about to be changed.
 OnActiveRowChange	Invoked when the active row is about to be changed.
 OnAddRow	Invoked when a row is about to be added to the grid.
 OnAddRowBatch	Invoked when a row is about to be added to the grid.
 OnClick	Invoked when a row, a column or a cell is clicked.
 OnClickCellButton	Invoked when a cell button is clicked.
 OnCollapseRow	Invoked when a row is about to be collapsed.
 OnColumnMove	Invoked when the column has been moved.
 OnColumnSizeChange	Invoked when the width of a column has been changed since the last postback.
 OnDbClick	Invoked when a row, a column or a cell is double clicked.
 OnDeleteRow	Invoked when a row is about to be deleted.
 OnDeleteRowBatch	Invoked when a row is about to be deleted.
 OnDemandLoad	Raises the DemandLoad event.
 OnExpandRow	Invoked when a row is about to be expanded.

 OnGroupColumn	Invoked when a column is about to be grouped.
 OnInitializeBand	Invoked when band needs to be configured during data binding. This event can be useful if your DataSource contains more than one column in the band which can be considered as a band of the next level. In this case it is recommended to set the Infragistics.WebUI.UltraWebGrid.UltraGridBand.ChildBandColumnName property of the band to insure that proper column is used for the next level band.
 OnInitializeFooter	Invoked when footers need to be configured during data binding.
 OnInitializeGroupByRow	Invoked when a group by row needs to be initialized during its creation.
 OnInitializeLayout	Invoked when the grid's layout needs to be initialized during data binding.
 OnInitializeRow	Overloaded.
 OnPageIndexChanged	Invoked when a new page of data is about to be displayed when paging is enabled.
 OnRowSizeChange	Invoked when the height of a row is being changed.
 OnSelectedCellsChange	Invoked when a cell or cells in the grid are about to be selected or unselected.
 OnSelectedColumnsChange	Invoked when a column or columns in the grid are about to be selected or unselected.
 OnSelectedRowsChange	Invoked when a row or rows in the grid are about to be selected or unselected.
 OnSortColumn	Overloaded.
 OnTemplatedColumnRestored	Invoked after a Templated column has been restored from Viewstate.
 OnUnGroupColumn	Invoked when a column is about to be ungrouped.
 OnUpdateCell	Invoked when a cell's value is about to be updated.
 OnUpdateCellBatch	Invoked when a cell is about to be updated.
 OnUpdateGrid	Invoked when the grid is about to be updated, meaning the value of a cell or cells changed, or rows were added or deleted.
 PerformGroupRows	Sorts all rows and creates the Group-By hierarchy when changes to the rows collection are made.
 RaisePostBackEvent	
 RaisePostDataChangedEvent	Parse the client updates and apply them to the proper rows.
 RenderBeginTag (Inherited from System.Web.UI.WebControls.WebControl)	Renders the HTML opening tag of the control into the specified writer. This method is used primarily by control developers.
 RenderControl (Inherited from System.Web.UI.Control)	Outputs server control content to a provided System.Web.UI.HtmlTextWriter object and stores tracing information about the control if tracing is enabled.
 RenderEndTag (Inherited from System.Web.UI.WebControls.WebControl)	Renders the HTML closing tag of the control into the specified writer. This method is used primarily by control developers.
 ResolveRowPath	Overloaded.
 ResolveUrl (Inherited from System.Web.UI.Control)	Converts a URL into one that is usable on the requesting client.











 SavePreset	Overloaded.
 SetRenderMethodDelegate (Inherited from System.Web.UI.Control)	

Public Events

 ActiveCellChange	Manages delegates of the type ActiveCellChangeEventHandler .
 ActiveRowChange	Manages delegates of the type ActiveRowChangeEventHandler .
 AddRow	Manages delegates of the type AddRowEventHandler .
 AddRowBatch	Manages delegates of the type AddRowBatchEventHandler .
 Click	Manages delegates of the type ClickEventHandler .
 ClickCellButton	Manages delegates of the type ClickCellButtonEventHandler .
 CollapseRow	Manages delegates of the type CollapseRowEventHandler .
 ColumnMove	Manages delegates of the type ColumnMoveEventHandler .
 ColumnSizeChange	Manages delegates of the type ColumnSizeChangeEventHandler .
 DataBinding (Inherited from System.Web.UI.Control)	
 DbClick	Manages delegates of the type DbClickEventHandler .
 DeleteRow	Manages delegates of the type DeleteRowEventHandler .
 DeleteRowBatch	Manages delegates of the type DeleteRowBatchEventHandler .
 DemandLoad	Occurs when a Row expand button has been clicked and its children need to be populated.
 Disposed (Inherited from System.Web.UI.Control)	
 ExpandRow	Manages delegates of the type ExpandRowEventHandler .
 GroupColumn	Manages delegates of the type GroupColumnEventHandler .
 Init (Inherited from System.Web.UI.Control)	
 InitializeBand	Manages delegates of the type InitializeBandEventHandler .
 InitializeFooter	Manages delegates of the type InitializeFooterEventHandler .
 InitializeGroupByRow	Manages delegates of the type InitializeGroupByRowEventHandler .
 InitializeLayout	Manages delegates of the type InitializeLayoutEventHandler .
 InitializeRow	Manages delegates of the type InitializeRowEventHandler .
 Load (Inherited from System.Web.UI.Control)	












 PageIndexChanged	Manages delegates of the type PageIndexChangedEventHandler .
 PreRender (Inherited from System.Web.UI.Control)	
 RowSizeChange	Manages delegates of the type RowSizeChangeEventHandler .
 SelectedCellsChange	Manages delegates of the type SelectedCellsChangeEventHandler .
 SelectedColumnsChange	Manages delegates of the type SelectedColumnsChangeEventHandler .
 SelectedRowsChange	Manages delegates of the type SelectedColumnsChangeEventHandler .
 SortColumn	Manages delegates of the type SortColumnEventHandler .
 TemplatedColumnRestored	
 UnGroupColumn	Manages delegates of the type UnGroupColumnEventHandler .
 Unload (Inherited from System.Web.UI.Control)	
 UpdateCell	Manages delegates of the type UpdateCellEventHandler .
 UpdateCellBatch	
 UpdateGrid	Manages delegates of the type UpdateGridEventHandler .

Protected Properties

 ChildControlsCreated (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control's child controls have been created.
 Context (Inherited from System.Web.UI.Control)	Gets the System.Web.HttpContext object associated with the server control for the current Web request.
 DataKeysArray (Inherited from System.Web.UI.WebControls.BaseDataList)	
 Events (Inherited from System.Web.UI.Control)	Gets a list of event handler delegates for the control. This property is read-only.
 HasChildViewState (Inherited from System.Web.UI.Control)	Gets a value indicating whether the current server control's child controls have any saved view-state settings.
 IsTrackingViewState (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control is saving changes to its view state.
 TagKey (Inherited from System.Web.UI.WebControls.WebControl)	Gets the System.Web.UI.HtmlTextWriterTag value that corresponds to this Web server control. This property is used primarily by control developers.
 TagName (Inherited from System.Web.UI.WebControls.WebControl)	Gets the name of the control tag. This property is used primarily by control developers.
 ViewState (Inherited from System.Web.UI.Control)	Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page.
 ViewStateIgnoresCase (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the System.Web.UI.StateBag object is case-insensitive.

Protected Methods

 AddAttributesToRender (Inherited from System.Web.UI.WebControls.WebControl)	Adds HTML attributes and styles that need to be rendered to the specified System.Web.UI.HtmlTextWriter . This method is used primarily by control developers.
 AddedControl (Inherited from System.Web.UI.Control)	Called after a control is added to the System.Web.UI.Control.Controls collection of another control.
 AddParsedSubObject (Inherited from System.Web.UI.WebControls.BaseDataList)	
 BuildProfileTree (Inherited from System.Web.UI.Control)	
 ClearChildViewState (Inherited from System.Web.UI.Control)	Deletes the view-state information for all the server control's child controls.
 CreateChildControls	Overridden.
 CreateControlCollection (Inherited from System.Web.UI.Control)	Creates a new System.Web.UI.ControlCollection object to hold the child controls (both literal and server) of the server control.
 CreateControlHierarchy	Overridden.
 CreateControlStyle (Inherited from System.Web.UI.WebControls.WebControl)	Creates the style object that is used internally by the System.Web.UI.WebControls.WebControl class to implement all style related properties. This method is used primarily by control developers.
 EnsureChildControls (Inherited from System.Web.UI.Control)	Determines whether the server control contains child controls. If it does not, it creates child controls. Determines whether the server control contains child controls. If it does not, it creates child controls.
 GroupRows	
 IsLiteralContent (Inherited from System.Web.UI.Control)	Determines if the server control holds only literal content.
 LoadViewState	Overridden. Called by the framework to load round-trip data into the grid.
 MapPathSecure (Inherited from System.Web.UI.Control)	Retrieves the physical path that a virtual path, either absolute or relative, maps to.
 OnBubbleEvent (Inherited from System.Web.UI.Control)	Determines whether the event for the server control is passed up the page's UI server control hierarchy.
 OnDataBinding (Inherited from System.Web.UI.WebControls.BaseDataList)	
 OnInit (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Init event.
 OnLoad (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Load event.
 OnPreRender (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.PreRender event.
 OnSelectedIndexChanged (Inherited from System.Web.UI.WebControls.BaseDataList)	Raises the System.Web.UI.WebControls.BaseDataList.SelectedIndexChanged event of a System.Web.UI.WebControls.BaseDataList . This allows you to create a custom handler for the event.
 OnUnload (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Unload event. Server controls should perform any final cleanup, such as closing files, closing database connections, and discarding objects, during this stage of the server control lifecycle.

 PrepareControlHierarchy	Overridden.
 RaiseBubbleEvent (Inherited from System.Web.UI.Control)	Assigns any sources of the event and its information to the control's parent.
 RemovedControl (Inherited from System.Web.UI.Control)	Called after a control is removed from the System.Web.UI.Control.Controls collection of another control.
 Render	Overridden. Render this control to the output parameter specified.
 RenderChildren (Inherited from System.Web.UI.Control)	Outputs the content of a server control's children to a provided System.Web.UI.HtmlTextWriter object, which writes the content to be rendered on the client.
 RenderContents (Inherited from System.Web.UI.WebControls.WebControl)	Renders the contents of the control into the specified writer. This method is used primarily by control developers.
 RenderDesignTime	
 SaveViewState	Overridden. Called by the framework to load round-trip data from the grid into the html ViewState variable.
 SortGroupRows	
 TrackViewState	Overridden.
 UnGroupRows	

See Also

[UltraWebGrid Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UpdateEventArgs Class

Event arguments that are passed to event handlers that involve cell updates.

For a list of all members of this type, see [UpdateEventArgs members](#).

Object Model

UpdateEventArgs

└─ **Grid (UltraWebGrid)**

Inheritance Hierarchy

[System.Object](#)

└─ [System.EventArgs](#)

└─ [Infragistics.WebUI.UltraWebGrid.UltraGridEventArgs](#)

└─ **Infragistics.WebUI.UltraWebGrid.UpdateEventArgs**

Syntax

```
[Visual Basic]
Public Class UpdateEventArgs
    Inherits UltraGridEventArgs
```

```
[C#]
public class UpdateEventArgs : UltraGridEventArgs
```

```
[JScript]
public class UpdateEventArgs extends UltraGridEventArgs
```



See Also

[UpdateEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

UpdateEventArgs Class Members

[UpdateEventArgs overview](#)

Public Properties

 Cancel (Inherited from UltraGridEventArgs)	Property that allows processing of the event to be canceled.
 Grid	A reference to the grid to which the event applies.

See Also

[UpdateEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ValidatorItem Class

For a list of all members of this type, see [ValidatorItem members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.ValidatorItem

Syntax

```
[Visual Basic]
Public Class ValidatorItem
    Implements IStateManager
```

```
[C#]
public class ValidatorItem : IStateManager
```

```
[JScript]
public class ValidatorItem implements IStateManager
```


See Also

[ValidatorItem Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)


ValidatorItem Class Members

[ValidatorItem overview](#)


Public Constructors

 ValidatorItem Constructor	Overloaded.
--	-------------


Public Properties

 Value	Name of the validator control.
--	--------------------------------




Public Methods

 ToString	Overridden.
---	-------------

Protected Properties

 ViewState	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.
--	--

Protected Methods

 LoadViewState	
 SaveViewState	
 TrackViewState	

See Also

[ValidatorItem Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ValidatorItemsCollection Class

Summary description for ValidatorItems.

For a list of all members of this type, see [ValidatorItemsCollection members](#).

Inheritance Hierarchy

[System.Object](#)

[System.Collections.ArrayList](#)

Infragistics.WebUI.UltraWebGrid.ValidatorItemsCollection

Syntax

```
[Visual Basic]
Public Class ValidatorItemsCollection
    Inherits ArrayList
    Implements IList, ICollection, IEnumerable, ICloneable, IStateManager
```

```
[C#]
public class ValidatorItemsCollection : ArrayList,
IList, ICollection, IEnumerable, ICloneable, IStateManager
```

```
[JScript]
public class ValidatorItemsCollection extends ArrayList implements
IList, ICollection, IEnumerable, ICloneable, IStateManager
```


See Also

[ValidatorItemsCollection Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)








ValidatorItemsCollection Class Members

[ValidatorItemsCollection overview](#)



Public Constructors








 ValidatorItemsCollection Constructor	Overloaded.
---	-------------

Public Properties


 Capacity (Inherited from System.Collections.ArrayList)	
 Count (Inherited from System.Collections.ArrayList)	
 IsFixedSize (Inherited from System.Collections.ArrayList)	
 IsReadOnly (Inherited from System.Collections.ArrayList)	
 IsSynchronized (Inherited from System.Collections.ArrayList)	
 Item (Inherited from System.Collections.ArrayList)	
 SyncRoot (Inherited from System.Collections.ArrayList)	

Public Methods




 Add	Overloaded.
 AddRange (Inherited from System.Collections.ArrayList)	
 BinarySearch (Inherited from System.Collections.ArrayList)	Overloaded.
 Clear (Inherited from System.Collections.ArrayList)	
 Clone (Inherited from System.Collections.ArrayList)	
 Contains	Overloaded.
 CopyTo	Overloaded.
 GetEnumerator (Inherited from System.Collections.ArrayList)	Overloaded.
 GetRange (Inherited from System.Collections.ArrayList)	
 IndexOf	Overloaded.
 Insert	Overloaded.
 InsertRange (Inherited from System.Collections.ArrayList)	
 LastIndexOf (Inherited from System.Collections.ArrayList)	Overloaded.
 Remove	Overloaded.

 RemoveAt (Inherited from System.Collections.ArrayList)	
 RemoveRange (Inherited from System.Collections.ArrayList)	
 Reverse (Inherited from System.Collections.ArrayList)	Overloaded.
 SetRange (Inherited from System.Collections.ArrayList)	
 Sort (Inherited from System.Collections.ArrayList)	Overloaded.
 ToArray (Inherited from System.Collections.ArrayList)	Overloaded.
 TrimToSize (Inherited from System.Collections.ArrayList)	

Protected Properties

 ViewState	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.
--	--

Protected Methods

 LoadViewState	
 SaveViewState	
 TrackViewState	

See Also

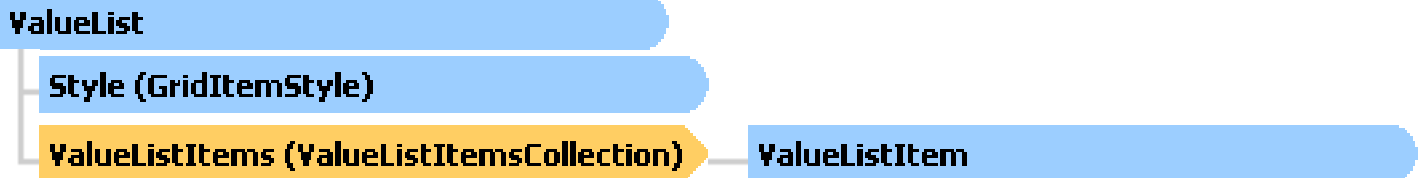
[ValidatorItemsCollection Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ValueList Class

The ValueList object maintains a list of values that the user can choose from when entering data. The ValueList is the common mechanism used to populate simple dropdown lists.

For a list of all members of this type, see [ValueList members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.KeyedObjectBase
Infragistics.WebUI.UltraWebGrid.ValueList

Syntax

```
[Visual Basic]  
Public Class ValueList  
    Inherits KeyedObjectBase  
    Implements IKeyedObject, IStateManager
```

```
[C#]  
public class ValueList : KeyedObjectBase, IKeyedObject, IStateManager
```

```
[JScript]  
public class ValueList extends KeyedObjectBase implements IKeyedObject, IStateManager
```

Remarks

The ValueList object maintains a collection of items that can be displayed as the contents of a dropdown selection box. The user can click on a dropdown list in their browser and select the data supplied by the ValueList, which will appear in the grid. ValueList objects are associated with a particular column in the grid by using the **ValueList** property of the Column object. Each cell of the column will display the value list when placed into edit mode. ValueList objects display one value as text to the user while storing another value in the datasource.


See Also

[ValueList Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)











ValueList Class Members

[ValueList overview](#)






Public Constructors

 ValueList Constructor	Overloaded.
--	-------------

Public Properties

 DataMember	Returns or sets a string of text that specifies the record set of the data source used to populate the ValueList object.
 DataSource	Returns or sets the data source that populates the items of the ValueList object.
 DisplayMember	Returns or sets a string of text that specifies the field of the table in the data source that will provide the item text for a ValueList object.
 DisplayStyle	Returns or sets a value that specifies whether DisplayText or DataValue of the ValueListItem objects are shown in the ValueList .
 HasStyle	Returns a Boolean value that determines whether the Style property is currently set to a GridItemStyle object.
 Key	Overridden. Returns or sets a string of text that uniquely identifies this ValueList within a Infragistics.WebUI.UltraWebGrid.ValueLists collection.
 Prompt	Appears in the first line of the value list and is selected by default if there is no matching item in the list.
 Style	Returns a reference to or sets a GridItemStyle object that determines how the ValueList is rendered on the client.
 ValueListItems	Returns a reference to a ValueListItemsCollection collection that contains a collection of ValueListItem objects.
 ValueMember	Returns or sets a string of text that specifies the field of the table in the data source that will provide the item values for a ValueList object.



Public Methods

 CopyFrom	Duplicates the properties of the specified ValueList into the instance of the ValueList class from which this method is invoked.
 DataBind	Overloaded. Binds the ValueList to the data source specified by the DataSource property.
 Reset	Resets all properties of the ValueList class to their default values.
 ResetKey (Inherited from KeyedObjectBase)	Sets the key back to null
 ShouldSerializeKey (Inherited from KeyedObjectBase)	Returns true if the key needs to be serialized (not null)



 [ToString](#)

Overridden. Returns a string representation of an [ValueList](#) object.

Protected Properties

 IsTrackingViewState	
 ViewState	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.

Protected Methods

 OnAddedToCollection (Inherited from KeyedObjectBase)	Called when this object is being added to the passed in collection. The default implementation sets the internal primaryCollection reference if it hasn't already been set
 OnRemovedFromCollection (Inherited from KeyedObjectBase)	Called when this object is being removed from the passed in collection. The default implementation nulls out the primaryCollection if the passed in collection matches

See Also

[ValueList Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ValueListItem Class

The ValueListItem object represents a single row within the dropdown ValueList collection. The object manages the information for displaying and updating the item within the browser.

For a list of all members of this type, see [ValueListItem members](#).

Object Model

ValueListItem

ValueList

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.KeyedObjectBase

Infragistics.WebUI.UltraWebGrid.ValueListItem

Syntax

```
[Visual Basic]
Public Class ValueListItem
    Inherits KeyedObjectBase
    Implements IKeyedObject, IStateManager
```

```
[C#]
public class ValueListItem : KeyedObjectBase, IKeyedObject, IStateManager
```

```
[JScript]
public class ValueListItem extends KeyedObjectBase implements IKeyedObject, IStateManager
```


See Also

[ValueListItem Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)






ValueListItem Class Members

[ValueListItem overview](#)






Public Constructors

 ValueListItem Constructor	Overloaded.
--	-------------


Public Properties

 DataValue	Returns or sets a value that specifies the value stored in the data source when this ValueListItem is selected by the user from a ValueList .
 DisplayText	Returns or sets a string of text that specifies the text displayed when this ValueListItem is selected by the user from a ValueList .
 Key	Overridden. Returns or sets a string of text that uniquely identifies this ValueListItem within a Infragistics.WebUI.UltraWebGrid.ValueListItems collection.
 Tag	Returns or sets an object that contains any additional data associated with this ValueListItem .
 ValueList	Returns a reference to or sets the ValueList object that contains this ValueListItem .



Public Methods

 CopyFrom	Duplicates the properties of the specified ValueListItem into the instance of the ValueListItem class from which this method is invoked.
 Reset	Resets all properties of the ValueListItem class to their default values.
 ResetKey (Inherited from KeyedObjectBase)	Sets the key back to null
 ShouldSerializeKey (Inherited from KeyedObjectBase)	Returns true if the key needs to be serialized (not null)
 ToString	Overridden. Returns a string representation of an ValueListItem object.

Protected Properties

 ViewState	Gets a dictionary of state information (StateBag) that allows you to save and restore the view state of a server control across multiple requests for the same page.
--	--

Protected Methods

 OnAddedToCollection (Inherited from KeyedObjectBase)	Called when this object is being added to the passed in collection. The default implementation sets the internal primaryCollection reference if it hasn't already been set
 OnRemovedFromCollection (Inherited from KeyedObjectBase)	Called when this object is being removed from the passed in collection. The default implementation nulls out the primaryCollection if the passed in collection matches

See Also

[ValueListItem Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

© 2004 • Infragistics, Inc.

ValueListItemsCollection Class

The collection of ValueListItem objects that make up a ValueList object. This collection manages the adding, inserting and removal of ValueListItem elements from the list.

For a list of all members of this type, see [ValueListItemsCollection members](#).

Object Model



Inheritance Hierarchy

[System.Object](#)

[System.Collections.CollectionBase](#)

[Infragistics.WebUI.UltraWebGrid.KeyedObjectCollectionBase](#)

[Infragistics.WebUI.UltraWebGrid.ValueListItemsCollection](#)

Syntax

```
[Visual Basic]
Public Class ValueListItemsCollection
    Inherits KeyedObjectCollectionBase
    Implements IList, ICollection, IEnumerable, IStateManager
```

```
[C#]
public class ValueListItemsCollection : KeyedObjectCollectionBase,
IList, ICollection, IEnumerable, IStateManager
```

```
[JScript]
public class ValueListItemsCollection extends KeyedObjectCollectionBase implements
IList, ICollection, IEnumerable, IStateManager
```


See Also

[ValueListItemsCollection Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)







ValueListItemCollection Class Members

[ValueListItemCollection overview](#)













Public Constructors

 ValueListItemCollection Constructor	
--	--

Public Properties


 All (Inherited from KeyedObjectCollectionBase)	The collection as an array of objects
 Count (Inherited from System.Collections.CollectionBase)	
 IsReadOnly	Overridden. Returns a Boolean value indicating whether the collection is read-only.
 Item	ValueListItem property indexer for the ValueListItem collection.
 Owner (Inherited from KeyedObjectCollectionBase)	Provides public access to the owning object of this collection
 ValueList	Returns a reference to or sets the ValueList object that contains this ValueListItem .

Public Methods













 Add	Overloaded.
 Clear (Inherited from System.Collections.CollectionBase)	Removes all the ValueListItem objects from the collection.
 Contains (Inherited from KeyedObjectCollectionBase)	Returns true if the collection contains this item
 CopyTo (Inherited from KeyedObjectCollectionBase)	Copies the items into the array
 Exists (Inherited from KeyedObjectCollectionBase)	Returns true if an object with this key is already in the collection. Note, if key is null or a zero length string this method returns false
 GetEnumerator	Returns an enumerator object for the collection.
 GetItem (Inherited from KeyedObjectCollectionBase)	Overloaded.
 IndexOf (Inherited from KeyedObjectCollectionBase)	Overloaded.
 Insert	Overloaded.
 Remove	Removes the specified ValueListItem object from the collection.
 RemoveAt (Inherited from System.Collections.CollectionBase)	Removes the ValueListItem object at the specified index from the collection.
 ValidateKeyDoesNotExist (Inherited from KeyedObjectCollectionBase)	Overloaded.

Protected Properties

 InnerList (Inherited from System.Collections.CollectionBase)	
---	--

 List (Inherited from System.Collections.CollectionBase)	
---	--

Protected Methods

 CreateArray (Inherited from KeyedObjectCollectionBase)	Virtual method used by the All 'get' method to create the array it returns.
 InternalAdd (Inherited from KeyedObjectCollectionBase)	Appends the object to the collection
 InternalClear (Inherited from KeyedObjectCollectionBase)	Clears the collection
 InternalInsert (Inherited from KeyedObjectCollectionBase)	Inserts an object into the collection
 InternalRemove (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 InternalRemoveAt (Inherited from KeyedObjectCollectionBase)	Removes an item from the collection
 OnClear (Inherited from System.Collections.CollectionBase)	
 OnClearComplete (Inherited from System.Collections.CollectionBase)	
 OnInsert (Inherited from System.Collections.CollectionBase)	
 OnInsertComplete (Inherited from System.Collections.CollectionBase)	
 OnRemove (Inherited from System.Collections.CollectionBase)	
 OnRemoveComplete (Inherited from System.Collections.CollectionBase)	
 OnSet (Inherited from System.Collections.CollectionBase)	
 OnSetComplete (Inherited from System.Collections.CollectionBase)	
 OnValidate (Inherited from System.Collections.CollectionBase)	

See Also

[ValueListItemsCollection Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

XmlClientWrite Class

Summary description for XmlClientWrite.

For a list of all members of this type, see [XmlClientWrite members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.XmlWrite

Infragistics.WebUI.UltraWebGrid.XmlClientWrite

Syntax

```
[Visual Basic]
Public Class XmlClientWrite
    Inherits XmlWrite
```

```
[C#]
public class XmlClientWrite : XmlWrite
```

```
[JScript]
public class XmlClientWrite extends XmlWrite
```

See Also

[XmlClientWrite Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)




XmlClientWrite Class Members

[XmlClientWrite overview](#)

Public Constructors

 XmlClientWrite Constructor	
---	--

Public Methods

 GetFilename (Inherited from XmlWrite)	
 SaveLayout	Overridden.
 SaveLayoutData (Inherited from XmlWrite)	

See Also

[XmlClientWrite Class](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ProcessRowParams Structure

Parameter-passing structure for Row exporting activities.

Inheritance Hierarchy

[System.Object](#)

[System.ValueType](#)

Infragistics.WebUI.UltraWebGrid.ProcessRowParams

Syntax

```
[Visual Basic]
Public Structure ProcessRowParams
    Inherits ValueType
```

```
[C#]
public struct ProcessRowParams : ValueType
```

```
[JScript]
In JScript, you can use structures in other assemblies, but you cannot define your own.
```

Remarks

This structure contains a set of flags governing export processing of the current [Row](#).




See Also

[ProcessRowParams Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#) | [IUltraGridExporter](#)




ProcessRowParams Structure Members

[ProcessRowParams overview](#)

Public Fields

 SkipDescendants	Specifies whether to skip the descendats of the current row.
 SkipSiblings	Specifies whether to skip sibling rows of the current row.
 TerminateExport	Specifies whether to terminate the export process. Current row will not be processed.

Public Methods

 Equals (Inherited from System.ValueType)	
 GetHashCode (Inherited from System.ValueType)	
 ToString (Inherited from System.ValueType)	

See Also

[ProcessRowParams Structure](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#) | [IUltraGridExporter](#)

ICanEditNavBar Interface

For a list of all members of this type, see [ICanEditNavBar members](#).

Syntax

```
[Visual Basic]  
Public Interface ICanEditNavBar
```

```
[C#]  
public interface ICanEditNavBar
```

```
[JScript]  
public interface ICanEditNavBar
```

See Also

[ICanEditNavBar Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

ICanEditNavBar Interface Members

[ICanEditNavBar overview](#)

Public Properties

 IsNavBarEditable	
---	--

Public Methods

 EditNavBar	
---	--

See Also

[ICanEditNavBar Interface](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

IControlEditorConsumer Interface

Interface that supports reuse of an Editor Control infrastructure for arbitrary components on a Web Form. For a list of all members of this type, see [IControlEditorConsumer members](#).

Syntax

```
[Visual Basic]  
Public Interface IControlEditorConsumer
```

```
[C#]  
public interface IControlEditorConsumer
```

```
[JScript]  
public interface IControlEditorConsumer
```


See Also

[IControlEditorConsumer Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)




IControlEditorConsumer Interface Members

[IControlEditorConsumer overview](#)

Public Properties

 BoundType	Retrieves the fully-qualified Type of the <i>target</i> instance.
--	--

Public Methods

 GetControls	Retrieves the collection of controls on the Web Form along with <i>target</i> .
 IsApplicable	Predicate method that identifies whether this <i>control</i> should be populated in the editor's dropdown listbox.
 OnValueChanged	Callback made when a Control ID for use as this <i>target's</i> editor has been selected in the design-time Property Editor's dropdown listbox.

See Also

[IControlEditorConsumer Interface](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

IPlugInConsumer Interface

Interface permitting registration of external plug-ins.

For a list of all members of this type, see [IPlugInConsumer members](#).

Syntax

```
[Visual Basic]  
Public Interface IPlugInConsumer
```

```
[C#]  
public interface IPlugInConsumer
```

```
[JScript]  
public interface IPlugInConsumer
```

Remarks

The UltraWebGrid control implements this interface to act as a consumer of content rendered by external plug-ins. This interface allows external plug-ins to register their conforming implementations with the UltraWebGrid.



See Also

[IPlugInConsumer Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

IPlugInConsumer Interface Members

[IPlugInConsumer overview](#)

Public Methods

 AddFactory	Provides an IPlugInFactory to UltraWebGrid to use when creating instances of external plug-ins.
 RemoveFactory	Removes a previously registered IPlugInFactory with UltraWebGrid to use when creating instances of external plug-ins.

See Also

[IPlugInConsumer Interface](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

IPlugInFactory Interface

Interface permitting implementations of plug-in interfaces to be created.

For a list of all members of this type, see [IPlugInFactory members](#).

Syntax

```
[Visual Basic]  
Public Interface IPlugInFactory
```

```
[C#]  
public interface IPlugInFactory
```

```
[JScript]  
public interface IPlugInFactory
```

Remarks

Plug-in providers must implement this in one of their components and supply it to an [IPlugInConsumer](#) through that Consumer's [AddFactory](#) method. It is important that the plug-in implementation remove itself


See Also

[IPlugInFactory Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

IPlugInFactory Interface Members

[IPlugInFactory overview](#)

Public Methods

 CreateExternalRenderer	Creates a concrete implementation of the IPlugInRender interface that can be used to insert HTML into that rendered by UltraWebGrid at specific points.
---	---

See Also

[IPlugInFactory Interface](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

IPlugInRender Interface

An interface that defines the services an external plug-in must provide to UltraWebGrid to inject customized HTML into the rendering.

For a list of all members of this type, see [IPlugInRender members](#).

Syntax

```
[Visual Basic]  
Public Interface IPlugInRender
```

```
[C#]  
public interface IPlugInRender
```

```
[JScript]  
public interface IPlugInRender
```

Remarks

Developers that want to inject custom HTML at one of the extensibility points defined by the [RenderLocation](#) enumeration must implement this interface and [IPlugInFactory](#) and then register their factory object with an [IPlugInConsumer](#) such as UltraWebGrid.

Developers who wish confirmation that their **IPlugInRender** implementation has been called but not used (because [Location](#) does not match the current rendering point), or that have resource-intensive implementations, are advised to additionally implement [Dispose](#) for end-of-life notification and clean-up.


See Also

[IPlugInRender Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)



IPlugInRender Interface Members

[IPlugInRender overview](#)

Public Properties

 Location	Point in UltraWebGrid's rendering at which this HTML is to be injected.
---	---

Public Methods

 Dispose	Disposes of any resources retained by the current IPlugInRender implementation.
 Render	Renders custom HTML content at the current Location in UltraWebGrid.

See Also

[IPlugInRender Interface](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

IResolveStyles Interface

Style resolution interface.

For a list of all members of this type, see [IResolveStyles members](#).

Syntax

```
[Visual Basic]  
Public Interface IResolveStyles
```

```
[C#]  
public interface IResolveStyles
```

```
[JScript]  
public interface IResolveStyles
```


See Also

[IResolveStyles Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)


IResolveStyles Interface Members

[IResolveStyles overview](#)

Public Properties

 Context	Contextual information about where the style resolution is being performed (for example, the header or footer of a column).
--	---

Public Methods

 Resolve	Overloaded.
--	-------------

See Also

[IResolveStyles Interface](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

IUltraGridExporter Interface

Interface for exporting UltraWebGrid into alternative formats.

For a list of all members of this type, see [IUltraGridExporter members](#).

Syntax

```
[Visual Basic]  
Public Interface IUltraGridExporter
```

```
[C#]  
public interface IUltraGridExporter
```

```
[JScript]  
public interface IUltraGridExporter
```

Remarks

Implementation classes that facilitate the transformation of [UltraWebGrid](#) into alternative formats via any row-wise methodology should implement these methods:

- BeginExport
ProcessRow
EndExport

Please see the documentation for these interface methods for further details on the requirements and responsibilities of an implementation class.




See Also

[IUltraGridExporter Members](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

IUltraGridExporter Interface Members

[IUltraGridExporter overview](#)

Public Methods

 BeginExport	Starts the export process into an alternate format.
 EndExport	Concludes the export processing.
 ProcessRow	Exports the Row instance supplied.

See Also

[IUltraGridExporter Interface](#) | [Infragistics.WebUI.UltraWebGrid Namespace](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace

[Inheritance Hierarchy](#)

Classes

Class	Description
BeginExportEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.BeginExport event.
CellExportedEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.CellExported event.
CellExportingEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.CellExporting event.
ColumnSet.ColumnIndex	This is a wrapper that can be used on integer indexes that represent spreadsheet columns.
EndExportEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.EndExport event.
ExcelExportCancelEventArgs	ExcelExportCancelEventArgs is a base class for cancellable event argument classes.
ExcelExportEventArgs	ExcelExportEventArgs is a base class for non-cancellable event argument classes.
ExcelExportInitializeRowEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.InitializeRow event.
HeaderCellExportedEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderCellExported event.
HeaderCellExportingEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderCellExporting event.
HeaderRowExportedEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderRowExported event.
HeaderRowExportingEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderRowExporting event.
InitializeColumnEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.InitializeColumn event.
RowExportedEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.RowExported event.
RowExportingEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.RowExporting event.

SummaryCellExportedEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryCellExported event.
SummaryCellExportingEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryCellExporting event.
SummaryRowExportedEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryRowExported event.
SummaryRowExportingEventArgs	Event parameters used for the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryRowExporting event.
UltraWebGridExcelExporter	The UltraGrid Excel Exporter control provides exporting functionality to Microsoft Excel 97 and later.

Interfaces

Interface	Description
IGenerateGridExportEvents	Interface describing the events developers may subscribe themselves to from the UltraWebGridExcelExporter.

Enumerations

Enumeration	Description
ExportCell.CellContext	
ExportMode	Listing of export modes that control the behavior of the export as it is experienced by end users.
HeaderTypes	An enumeration classifying header types used in export-related events concerning headers.

Delegates

Delegate	Description
BeginExportEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.BeginExport event.
CellExportedEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.CellExported event.
CellExportingEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.CellExporting event.
EndExportEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.EndExport event.
ExcelExporterImpl.ProcessRowDelegate	
FireExportEvent	
HeaderCellExportedEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderCellExported event.

HeaderCellExportingEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderCellExporting event.
HeaderRowExportedEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderRowExported event.
HeaderRowExportingEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderRowExporting event.
IndentChangedEventHandler	
InitializeColumnEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.InitializeColumn event.
InitializeRowEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.InitializeRow event.
RowExportedEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.RowExported event.
RowExportingEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.RowExporting event.
SummaryCellExportedEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryCellExported event.
SummaryCellExportingEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryCellExporting event.
SummaryRowExportedEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryRowExported event.
SummaryRowExportingEventHandler	Delegate for handling the Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryRowExporting event.

See Also

[Infragistics.WebUI.UltraWebGrid.ExcelExport.v3.1 Assembly](#)

BeginExportEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.BeginExport** event.

For a list of all members of this type, see [BeginExportEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.BeginExportEventArgs

Syntax

```
[Visual Basic]
Public Class BeginExportEventArgs
    Inherits ExcelExportEventArgs
```

```
[C#]
public class BeginExportEventArgs : ExcelExportEventArgs
```

```
[JScript]
public class BeginExportEventArgs extends ExcelExportEventArgs
```








See Also

[BeginExportEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

BeginExportEventArgs Class Members

[BeginExportEventArgs overview](#)

Public Properties

 CurrentColumnIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting column in Excel worksheet.
 CurrentOutlineLevel (Inherited from ExcelExportEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting row in Excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportEventArgs)	Current exporting worksheet.
 Layout	Grid layout.
 Rows	Topmost band's rows collection.
 Workbook (Inherited from ExcelExportEventArgs)	Exporting workbook.

See Also

[BeginExportEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

CellExportedEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.CellExported** event.

For a list of all members of this type, see [CellExportedEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.CellExportedEventArgs

Syntax

```
[Visual Basic]
Public Class CellExportedEventArgs
    Inherits ExcelExportEventArgs
```

```
[C#]
public class CellExportedEventArgs : ExcelExportEventArgs
```

```
[JScript]
public class CellExportedEventArgs extends ExcelExportEventArgs
```









See Also

[CellExportedEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

CellExportedEventArgs Class Members

[CellExportedEventArgs overview](#)

Public Properties

 CurrentColumnIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting column in Excel worksheet.
 CurrentOutlineLevel (Inherited from ExcelExportEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting row in Excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportEventArgs)	Current exporting worksheet.
 GridColumn	Associated grid column.
 GridRow	Grid row containing the cell.
 Value	Grid cell value.
 Workbook (Inherited from ExcelExportEventArgs)	Exporting workbook.

See Also

[CellExportedEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

CellExportingEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.CellExporting** event.

For a list of all members of this type, see [CellExportingEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[System.ComponentModel.CancelEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportCancelEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.CellExportingEventArgs

Syntax

```
[Visual Basic]
Public Class CellExportingEventArgs
    Inherits ExcelExportCancelEventArgs
```

```
[C#]
public class CellExportingEventArgs : ExcelExportCancelEventArgs
```

```
[JScript]
public class CellExportingEventArgs extends ExcelExportCancelEventArgs
```










See Also

[CellExportingEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

CellExportingEventArgs Class Members

[CellExportingEventArgs overview](#)

Public Properties

 Cancel (Inherited from System.ComponentModel.CancelEventArgs)	Gets or sets a value indicating whether the event should be canceled.
 CurrentColumnIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting column in excel worksheet..
 CurrentOutlineLevel (Inherited from ExcelExportCancelEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting row in excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportCancelEventArgs)	Current exporting worksheet.
 GridColumn	Associated grid column.
 GridRow	Grid row containing the cell.
 Value	Grid cell value.
 Workbook (Inherited from ExcelExportCancelEventArgs)	Exporting workbook.

See Also

[CellExportingEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

ColumnSet.ColumnIndex Class

This is a wrapper that can be used on integer indexes that represent spreadsheet columns.
For a list of all members of this type, see [ColumnSet.ColumnIndex members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.ColumnSet.ColumnIndex

Syntax

```
[Visual Basic]  
Public Class ColumnSet.ColumnIndex
```

```
[C#]  
public class ColumnSet.ColumnIndex
```

```
[JScript]  
public class ColumnSet.ColumnIndex
```

Remarks

Wrapping a column index in this class is necessary to make use of the overloaded [ColumnSet](#) indexer that returns the appropriate [ColumnDesc](#) for a given **WorksheetColumn**.

See Also

[ColumnSet.ColumnIndex Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

ColumnSet.ColumnIndex Class Members

[ColumnSet.ColumnIndex overview](#)

Public Constructors

 ColumnSet.ColumnIndex Constructor	
--	--

Public Properties

 ID	
---	--

See Also

[ColumnSet.ColumnIndex Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

EndExportEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.EndExport** event.

For a list of all members of this type, see [EndExportEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.EndExportEventArgs

Syntax

```
[Visual Basic]
Public Class EndExportEventArgs
    Inherits ExcelExportEventArgs
```

```
[C#]
public class EndExportEventArgs : ExcelExportEventArgs
```

```
[JScript]
public class EndExportEventArgs extends ExcelExportEventArgs
```







See Also

[EndExportEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

EndExportEventArgs Class Members

[EndExportEventArgs overview](#)

Public Properties

 Cancelled	True if the exporting process has been cancelled.
 CurrentColumnIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting column in Excel worksheet.
 CurrentOutlineLevel (Inherited from ExcelExportEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting row in Excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportEventArgs)	Current exporting worksheet.
 Workbook (Inherited from ExcelExportEventArgs)	Exporting workbook.

See Also

[EndExportEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

ExcelExportCancelEventArgs Class

ExcelExportCancelEventArgs is a base class for cancellable event argument classes.

For a list of all members of this type, see [ExcelExportCancelEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[System.ComponentModel.CancelEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportCancelEventArgs

[Infragistics.WebUI.UltraWebGrid.ExcelExport.SummaryRowExportingEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.SummaryCellExportingEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.RowExportingEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.HeaderRowExportingEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.HeaderCellExportingEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.CellExportingEventArgs](#)

Syntax

[Visual Basic]

```
Public Class ExcelExportCancelEventArgs  
    Inherits CancelEventArgs
```

[C#]

```
public class ExcelExportCancelEventArgs : CancelEventArgs
```

[JScript]

```
public class ExcelExportCancelEventArgs extends CancelEventArgs
```

Remarks

Many events occur in before/after pairings where the before event permits developers a chance to pre-empt an activity that the event provides notice to them of. When developers have set the [Cancel](#) property in their event handlers subscribed to cancellable 'before' events to **true**, the corresponding 'after' event will not fire because the activity will have been pre-empted.







See Also

[ExcelExportCancelEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

ExcelExportCancelEventArgs Class Members

[ExcelExportCancelEventArgs overview](#)

Public Properties

 Cancel (Inherited from System.ComponentModel.CancelEventArgs)	Gets or sets a value indicating whether the event should be canceled.
 CurrentColumnIndex	Zero-based index of current exporting column in excel worksheet..
 CurrentOutlineLevel	Current outline level used for grouping.
 CurrentRowIndex	Zero-based index of current exporting row in excel worksheet.
 CurrentWorksheet	Current exporting worksheet.
 Workbook	Exporting workbook.

See Also

[ExcelExportCancelEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

ExcelExportEventArgs Class

ExcelExportEventArgs is a base class for non-cancellable event argument classes.

For a list of all members of this type, see [ExcelExportEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportEventArgs

[Infragistics.WebUI.UltraWebGrid.ExcelExport.SummaryRowExportedEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.SummaryCellExportedEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.RowExportedEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.HeaderRowExportedEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.HeaderCellExportedEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportInitializeRowEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.EndExportEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.CellExportedEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.BeginExportEventArgs](#)

Syntax

```
[Visual Basic]
Public Class ExcelExportEventArgs
    Inherits EventArgs
```

```
[C#]
public class ExcelExportEventArgs : EventArgs
```

```
[JScript]
public class ExcelExportEventArgs extends EventArgs
```






See Also

[ExcelExportEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

ExcelExportEventArgs Class Members

[ExcelExportEventArgs overview](#)

Public Properties

 CurrentColumnIndex	Zero-based index of current exporting column in Excel worksheet.
 CurrentOutlineLevel	Current outline level used for grouping.
 CurrentRowIndex	Zero-based index of current exporting row in Excel worksheet.
 CurrentWorksheet	Current exporting worksheet.
 Workbook	Exporting workbook.

See Also

[ExcelExportEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

ExcelExportInitializeRowEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.InitializeRow** event.

For a list of all members of this type, see [ExcelExportInitializeRowEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportInitializeRowEventArgs

Syntax

```
[Visual Basic]
Public Class ExcelExportInitializeRowEventArgs
    Inherits ExcelExportEventArgs
```

```
[C#]
public class ExcelExportInitializeRowEventArgs : ExcelExportEventArgs
```

```
[JScript]
public class ExcelExportInitializeRowEventArgs extends ExcelExportEventArgs
```











See Also

[ExcelExportInitializeRowEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

ExcelExportInitializeRowEventArgs Class Members

[ExcelExportInitializeRowEventArgs overview](#)

Public Properties

 CurrentColumnIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting column in Excel worksheet.
 CurrentOutlineLevel (Inherited from ExcelExportEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting row in Excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportEventArgs)	Current exporting worksheet.
 Row	Grid row.
 SkipDescendants	Specifies whether to skip the descendants of the current row.
 SkipRow	Specifies whether to skip the current row.
 SkipSiblings	Specifies whether to skip sibling rows of the current row.
 TerminateExport	Specifies whether to terminate the export process. Current row will not be processed.
 Workbook (Inherited from ExcelExportEventArgs)	Exporting workbook.

See Also

[ExcelExportInitializeRowEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

HeaderCellExportedEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderCellExported** event.

For a list of all members of this type, see [HeaderCellExportedEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.HeaderCellExportedEventArgs

Syntax

```
[Visual Basic]
Public Class HeaderCellExportedEventArgs
    Inherits ExcelExportEventArgs
```

```
[C#]
public class HeaderCellExportedEventArgs : ExcelExportEventArgs
```

```
[JScript]
public class HeaderCellExportedEventArgs extends ExcelExportEventArgs
```








See Also

[HeaderCellExportedEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

HeaderCellExportedEventArgs Class Members

[HeaderCellExportedEventArgs overview](#)

Public Properties

 CurrentColumnIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting column in Excel worksheet.
 CurrentOutlineLevel (Inherited from ExcelExportEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting row in Excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportEventArgs)	Current exporting worksheet.
 GridRow	Associated grid row.
 HeaderType	Type of header.
 Workbook (Inherited from ExcelExportEventArgs)	Exporting workbook.

See Also

[HeaderCellExportedEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

HeaderCellExportingEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderCellExporting** event.

For a list of all members of this type, see [HeaderCellExportingEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[System.ComponentModel.CancelEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportCancelEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.HeaderCellExportingEventArgs

Syntax

```
[Visual Basic]
Public Class HeaderCellExportingEventArgs
    Inherits ExcelExportCancelEventArgs
```

```
[C#]
public class HeaderCellExportingEventArgs : ExcelExportCancelEventArgs
```

```
[JScript]
public class HeaderCellExportingEventArgs extends ExcelExportCancelEventArgs
```









See Also

[HeaderCellExportingEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

HeaderCellExportingEventArgs Class Members

[HeaderCellExportingEventArgs overview](#)

Public Properties

 Cancel (Inherited from System.ComponentModel.CancelEventArgs)	Gets or sets a value indicating whether the event should be canceled.
 CurrentColumnIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting column in excel worksheet..
 CurrentOutlineLevel (Inherited from ExcelExportCancelEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting row in excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportCancelEventArgs)	Current exporting worksheet.
 GridRow	Associated grid row.
 HeaderType	Type of header.
 Workbook (Inherited from ExcelExportCancelEventArgs)	Exporting workbook.

See Also

[HeaderCellExportingEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

HeaderRowExportedEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderRowExported** event.

For a list of all members of this type, see [HeaderRowExportedEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.HeaderRowExportedEventArgs

Syntax

```
[Visual Basic]
Public Class HeaderRowExportedEventArgs
    Inherits ExcelExportEventArgs
```

```
[C#]
public class HeaderRowExportedEventArgs : ExcelExportEventArgs
```

```
[JScript]
public class HeaderRowExportedEventArgs extends ExcelExportEventArgs
```









See Also

[HeaderRowExportedEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

HeaderRowExportedEventArgs Class Members

[HeaderRowExportedEventArgs overview](#)

Public Properties

 Band	The band associated with these headers.
 CurrentColumnIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting column in Excel worksheet.
 CurrentOutlineLevel (Inherited from ExcelExportEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting row in Excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportEventArgs)	Current exporting worksheet.
 GridRow	The row associated with these headers.
 HeaderType	The classification of this header.
 Workbook (Inherited from ExcelExportEventArgs)	Exporting workbook.

See Also

[HeaderRowExportedEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

HeaderRowExportingEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.HeaderRowExporting** event.

For a list of all members of this type, see [HeaderRowExportingEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[System.ComponentModel.CancelEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportCancelEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.HeaderRowExportingEventArgs

Syntax

```
[Visual Basic]
Public Class HeaderRowExportingEventArgs
    Inherits ExcelExportCancelEventArgs
```

```
[C#]
public class HeaderRowExportingEventArgs : ExcelExportCancelEventArgs
```

```
[JScript]
public class HeaderRowExportingEventArgs extends ExcelExportCancelEventArgs
```










See Also

[HeaderRowExportingEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

HeaderRowExportingEventArgs Class Members

[HeaderRowExportingEventArgs overview](#)

Public Properties

 Band	UltraGridBand associated with headers.
 Cancel (Inherited from System.ComponentModel.CancelEventArgs)	Gets or sets a value indicating whether the event should be canceled.
 CurrentColumnIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting column in excel worksheet..
 CurrentOutlineLevel (Inherited from ExcelExportCancelEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting row in excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportCancelEventArgs)	Current exporting worksheet.
 GridRow	UltraGridRow associated with headers.
 HeaderType	Type of header.
 Workbook (Inherited from ExcelExportCancelEventArgs)	Exporting workbook.

See Also

[HeaderRowExportingEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

InitializeColumnEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.InitializeColumn** event.

For a list of all members of this type, see [InitializeColumnEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.InitializeColumnEventArgs

Syntax

```
[Visual Basic]
Public Class InitializeColumnEventArgs
    Inherits EventArgs
```

```
[C#]
public class InitializeColumnEventArgs : EventArgs
```

```
[JScript]
public class InitializeColumnEventArgs extends EventArgs
```




See Also

[InitializeColumnEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

InitializeColumnEventArgs Class Members

[InitializeColumnEventArgs overview](#)

Public Properties

 Column	Reference to a column.
 ExcelFormatString	Microsoft Excel-specific format string.
 FormatString	Format string used in .NET Framework.

See Also

[InitializeColumnEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

RowExportedEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.RowExported** event.

For a list of all members of this type, see [RowExportedEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.RowExportedEventArgs

Syntax

```
[Visual Basic]
Public Class RowExportedEventArgs
    Inherits ExcelExportEventArgs
```

```
[C#]
public class RowExportedEventArgs : ExcelExportEventArgs
```

```
[JScript]
public class RowExportedEventArgs extends ExcelExportEventArgs
```







See Also

[RowExportedEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

RowExportedEventArgs Class Members

[RowExportedEventArgs overview](#)

Public Properties

 CurrentColumnIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting column in Excel worksheet.
 CurrentOutlineLevel (Inherited from ExcelExportEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting row in Excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportEventArgs)	Current exporting worksheet.
 GridRow	Reference to row being exported.
 Workbook (Inherited from ExcelExportEventArgs)	Exporting workbook.

See Also

[RowExportedEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

RowExportingEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter**. **RowExporting** event.

For a list of all members of this type, see [RowExportingEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[System.ComponentModel.CancelEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportCancelEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.RowExportingEventArgs

Syntax

```
[Visual Basic]
Public Class RowExportingEventArgs
    Inherits ExcelExportCancelEventArgs
```

```
[C#]
public class RowExportingEventArgs : ExcelExportCancelEventArgs
```

```
[JScript]
public class RowExportingEventArgs extends ExcelExportCancelEventArgs
```








See Also

[RowExportingEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

RowExportingEventArgs Class Members

[RowExportingEventArgs overview](#)

Public Properties

 Cancel (Inherited from System.ComponentModel.CancelEventArgs)	Gets or sets a value indicating whether the event should be canceled.
 CurrentColumnIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting column in excel worksheet..
 CurrentOutlineLevel (Inherited from ExcelExportCancelEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting row in excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportCancelEventArgs)	Current exporting worksheet.
 GridRow	Reference to row being exported.
 Workbook (Inherited from ExcelExportCancelEventArgs)	Exporting workbook.

See Also

[RowExportingEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

SummaryCellExportedEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryCellExported** event.

For a list of all members of this type, see [SummaryCellExportedEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.SummaryCellExportedEventArgs

Syntax

```
[Visual Basic]
Public Class SummaryCellExportedEventArgs
    Inherits ExcelExportEventArgs
```

```
[C#]
public class SummaryCellExportedEventArgs : ExcelExportEventArgs
```

```
[JScript]
public class SummaryCellExportedEventArgs extends ExcelExportEventArgs
```







See Also

[SummaryCellExportedEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

SummaryCellExportedEventArgs Class Members

[SummaryCellExportedEventArgs overview](#)

Public Properties

 CurrentColumnIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting column in Excel worksheet.
 CurrentOutlineLevel (Inherited from ExcelExportEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting row in Excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportEventArgs)	Current exporting worksheet.
 SummaryLevel	Summary level.
 Workbook (Inherited from ExcelExportEventArgs)	Exporting workbook.

See Also

[SummaryCellExportedEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

SummaryCellExportingEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryCellExporting** event.

For a list of all members of this type, see [SummaryCellExportingEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[System.ComponentModel.CancelEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportCancelEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.SummaryCellExportingEventArgs

Syntax

```
[Visual Basic]
Public Class SummaryCellExportingEventArgs
    Inherits ExcelExportCancelEventArgs
```

```
[C#]
public class SummaryCellExportingEventArgs : ExcelExportCancelEventArgs
```

```
[JScript]
public class SummaryCellExportingEventArgs extends ExcelExportCancelEventArgs
```








See Also

[SummaryCellExportingEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

SummaryCellExportingEventArgs Class Members

[SummaryCellExportingEventArgs overview](#)

Public Properties

 Cancel (Inherited from System.ComponentModel.CancelEventArgs)	Gets or sets a value indicating whether the event should be canceled.
 CurrentColumnIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting column in excel worksheet..
 CurrentOutlineLevel (Inherited from ExcelExportCancelEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting row in excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportCancelEventArgs)	Current exporting worksheet.
 SummaryLevel	Summary level.
 Workbook (Inherited from ExcelExportCancelEventArgs)	Exporting workbook.

See Also

[SummaryCellExportingEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

SummaryRowExportedEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryRowExported** event.

For a list of all members of this type, see [SummaryRowExportedEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.SummaryRowExportedEventArgs

Syntax

```
[Visual Basic]
Public Class SummaryRowExportedEventArgs
    Inherits ExcelExportEventArgs
```

```
[C#]
public class SummaryRowExportedEventArgs : ExcelExportEventArgs
```

```
[JScript]
public class SummaryRowExportedEventArgs extends ExcelExportEventArgs
```







See Also

[SummaryRowExportedEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

SummaryRowExportedEventArgs Class Members

[SummaryRowExportedEventArgs overview](#)

Public Properties

 CurrentColumnIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting column in Excel worksheet.
 CurrentOutlineLevel (Inherited from ExcelExportEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportEventArgs)	Zero-based index of current exporting row in Excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportEventArgs)	Current exporting worksheet.
 SummaryLevel	Summary level.
 Workbook (Inherited from ExcelExportEventArgs)	Exporting workbook.

See Also

[SummaryRowExportedEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

SummaryRowExportingEventArgs Class

Event parameters used for the **Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraGridExcelExporter.SummaryRowExporting** event.

For a list of all members of this type, see [SummaryRowExportingEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

[System.ComponentModel.CancelEventArgs](#)

[Infragistics.WebUI.UltraWebGrid.ExcelExport.ExcelExportCancelEventArgs](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.SummaryRowExportingEventArgs

Syntax

```
[Visual Basic]
Public Class SummaryRowExportingEventArgs
    Inherits ExcelExportCancelEventArgs
```

```
[C#]
public class SummaryRowExportingEventArgs : ExcelExportCancelEventArgs
```

```
[JScript]
public class SummaryRowExportingEventArgs extends ExcelExportCancelEventArgs
```








See Also

[SummaryRowExportingEventArgs Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

SummaryRowExportingEventArgs Class Members

[SummaryRowExportingEventArgs overview](#)

Public Properties

 Cancel (Inherited from System.ComponentModel.CancelEventArgs)	Gets or sets a value indicating whether the event should be canceled.
 CurrentColumnIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting column in excel worksheet..
 CurrentOutlineLevel (Inherited from ExcelExportCancelEventArgs)	Current outline level used for grouping.
 CurrentRowIndex (Inherited from ExcelExportCancelEventArgs)	Zero-based index of current exporting row in excel worksheet.
 CurrentWorksheet (Inherited from ExcelExportCancelEventArgs)	Current exporting worksheet.
 SummaryLevel	Summary level.
 Workbook (Inherited from ExcelExportCancelEventArgs)	Exporting workbook.

See Also

[SummaryRowExportingEventArgs Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

UltraWebGridExcelExporter Class

The UltraGrid Excel Exporter control provides exporting functionality to Microsoft Excel 97 and later.

For a list of all members of this type, see [UltraWebGridExcelExporter members](#).

Inheritance Hierarchy

[System.Object](#)

[System.Web.UI.Control](#)

Infragistics.WebUI.UltraWebGrid.ExcelExport.UltraWebGridExcelExporter

Syntax

```
[Visual Basic]
Public Class UltraWebGridExcelExporter
    Inherits Control
    Implements
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, IUltraLicensedComponent, IGenerateGridExportEvents
```

```
[C#]
public class UltraWebGridExcelExporter : Control,
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, IUltraLicensedComponent, IGenerateGridExportEvents
```

```
[JScript]
public class UltraWebGridExcelExporter extends Control implements
IComponent, IDisposable, IParserAccessor, IDataBindingsAccessor, IUltraLicensedComponent, IGenerateGridExportEvents
```

Remarks

Developers can use the export features of this control by dragging it onto a Web Form containing an **UltraWebGrid** control that they wish to export to Microsoft Excel 97 or later format (also known as Binary Interchange File Format, or BIFF, revision 8).

This control is non-visual in nature, meaning it does not render any HTML to the client browser. An export is performed by calling the [Export](#) method on an UltraWebGrid instance. At that time, the exporter can take control of the **Page** such that other controls on the page do not render, and it streams the exported data to the client's browser.

Nevertheless, developers can customize several facets of the exporter's processing declaratively from within the Properties Editor. Developers should set the following behavioral properties that govern the parameters of the export operation:

ExportMode	Determines whether the exported spreadsheet should attempt to open InBrowser with in-place activation, if the client supports it, or as a plain file Download . Developers may also choose a Custom mode, in which their web application is responsible for using the Export method and other logic to deliver the spreadsheet in a nontraditional manner.
DownloadName	Customizes the name of the file that will be created on the client's file system (even when opening in-place, the client's browser will cache the spreadsheet file locally using this filename).
WorksheetName	Customizes the name of the active worksheet that appears on the tab bar beneath the spreadsheet.
ExcelStartCol ExcelStartRow	These properties direct the export operation to take place at a location other than the top-left corner of the spreadsheet.

It is also possible to specify or default these parameters at run-time in the arguments of the [Export](#) method, which is usually called as part of an event handler within the web application (for instance, when the end user clicks a **Button** control to initiate the download of an exported spreadsheet).

During the export, a diverse event model is exposed that permits developers to customize the appearance and formatting of the exported spreadsheet. In some situations, event handling may be necessary for advanced web applications to mediate .NET format specifiers into Microsoft Excel formats during the export, or achieve greater styling of the final product's appearance than is possible through the automated conversion of a web styles into Excel.


See Also

[UltraWebGridExcelExporter Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

UltraWebGridExcelExporter Class Members

[UltraWebGridExcelExporter overview](#)









Public Constructors

 UltraWebGridExcelExporter Constructor	Overloaded.
--	-------------




















Public Properties



 About	
 BindingContainer (Inherited from System.Web.UI.Control)	
 ClientID (Inherited from System.Web.UI.Control)	Gets the server control identifier generated by ASP.NET.
 Controls (Inherited from System.Web.UI.Control)	Gets a System.Web.UI.ControlCollection object that represents the child controls for a specified server control in the UI hierarchy.
 DownloadName	Filename given to the Workbook downloaded to the client during export.
 EnableViewState (Inherited from System.Web.UI.Control)	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client.
 ExcelStartColumn	Starting Excel column where grid data is exported into the document.
 ExcelStartRow	Starting Excel row where grid data is exported into the document.
 ExportMode	Determines what export behavior the control takes automatically when Export has been called.
 ID (Inherited from System.Web.UI.Control)	Gets or sets the programmatic identifier assigned to the server control.
 NamingContainer (Inherited from System.Web.UI.Control)	Gets a reference to the server control's naming container, which creates a unique namespace for differentiating between server controls with the same System.Web.UI.Control.ID property value.
 Page (Inherited from System.Web.UI.Control)	Gets a reference to the System.Web.UI.Page instance that contains the server control.
 Parent (Inherited from System.Web.UI.Control)	Gets a reference to the server control's parent control in the page control hierarchy.
 Site (Inherited from System.Web.UI.Control)	Gets information about the Web site to which the server control belongs.
 TemplateSourceDirectory (Inherited from System.Web.UI.Control)	Gets the virtual directory of the System.Web.UI.Page or System.Web.UI.UserControl that contains the current server control.
 UniqueID (Inherited from System.Web.UI.Control)	Gets the unique, hierarchically-qualified identifier for the server control.
 Visible	Overridden.
 WorksheetName	Name given to the Worksheet in an exported Workbook.

Public Methods









 DataBind (Inherited from System.Web.UI.Control)	Binds a data source to the invoked server control and all its child controls.
 Dispose (Inherited from System.Web.UI.Control)	Enables a server control to perform final clean up before it is released from memory.
 Export	Overloaded.
 FindControl (Inherited from System.Web.UI.Control)	Overloaded.
 HasControls (Inherited from System.Web.UI.Control)	Determines if the server control contains any child controls.
 RenderControl (Inherited from System.Web.UI.Control)	Outputs server control content to a provided System.Web.UI.HtmlTextWriter object and stores tracing information about the control if tracing is enabled.
 ResolveUrl (Inherited from System.Web.UI.Control)	Converts a URL into one that is usable on the requesting client.
 SetRenderMethodDelegate (Inherited from System.Web.UI.Control)	

Public Events













 BeginExport	Occurs before the UltraWebGrid export starts.
 CellExported	Occurs after an UltraWebGrid cell is exported to Microsoft Excel format.
 CellExporting	Occurs before an UltraWebGrid cell is exported to Microsoft Excel format.
 DataBinding	
 Disposed (Inherited from System.Web.UI.Control)	
 EndExport	Occurs after the UltraWebGrid export is finished.
 HeaderCellExported	Occurs after header cell is exported to Microsoft Excel format.
 HeaderCellExporting	Occurs before header cell is exported to Microsoft Excel format.
 HeaderRowExported	Occurs after header row is exported to Microsoft Excel format.
 HeaderRowExporting	Occurs before header row is exported to Microsoft Excel format.
 Init (Inherited from System.Web.UI.Control)	
 InitializeColumn	
 InitializeRow	Occurs when each UltraWebGrid row is initialized.
 Load (Inherited from System.Web.UI.Control)	
 PreRender (Inherited from System.Web.UI.Control)	
 RowExported	Occurs after an UltraWebGrid row is exported to Microsoft Excel format.
 RowExporting	Occurs before an UltraWebGrid row is exported to Microsoft Excel format.
 SummaryCellExported	Occurs after summary cell is exported to Microsoft Excel format.
 SummaryCellExporting	Occurs before summary cell is exported to Microsoft Excel format.
 SummaryRowExported	Occurs after summary row is exported to Microsoft Excel format.

 SummaryRowExporting	Occurs before summary row is exported to Microsoft Excel format.
 Unload (Inherited from System.Web.UI.Control)	



Protected Properties

 ChildControlsCreated (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control's child controls have been created.
 Context (Inherited from System.Web.UI.Control)	Gets the System.Web.HttpContext object associated with the server control for the current Web request.
 Events (Inherited from System.Web.UI.Control)	Gets a list of event handler delegates for the control. This property is read-only.
 ExportCalled	Indicates whether an Export method has been called.
 HasChildViewState (Inherited from System.Web.UI.Control)	Gets a value indicating whether the current server control's child controls have any saved view-state settings.
 IsTrackingViewState (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control is saving changes to its view state.
 ViewState (Inherited from System.Web.UI.Control)	Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page.
 ViewStateIgnoresCase (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the System.Web.UI.StateBag object is case-insensitive.

Protected Methods

 AddedControl (Inherited from System.Web.UI.Control)	Called after a control is added to the System.Web.UI.Control.Controls collection of another control.
 AddParsedSubObject (Inherited from System.Web.UI.Control)	Notifies the server control that an element, either XML or HTML, was parsed, and adds the element to the server control's System.Web.UI.ControlCollection object.
 BuildProfileTree (Inherited from System.Web.UI.Control)	
 ClearChildViewState (Inherited from System.Web.UI.Control)	Deletes the view-state information for all the server control's child controls.
 CreateChildControls (Inherited from System.Web.UI.Control)	Notifies server controls that use composition-based implementation to create any child controls they contain in preparation for posting back or rendering.
 CreateControlCollection (Inherited from System.Web.UI.Control)	Creates a new System.Web.UI.ControlCollection object to hold the child controls (both literal and server) of the server control.
 EnsureChildControls (Inherited from System.Web.UI.Control)	Determines whether the server control contains child controls. If it does not, it creates child controls.
 IsLiteralContent (Inherited from System.Web.UI.Control)	Determines if the server control holds only literal content.
 LoadViewState (Inherited from System.Web.UI.Control)	Restores view-state information from a previous page request that was saved by the System.Web.UI.Control.SaveViewState method.
 MapPathSecure (Inherited from System.Web.UI.Control)	Retrieves the physical path that a virtual path, either absolute or relative, maps to.
 OnBeginExport	Called before grid export starts.
 OnBubbleEvent (Inherited from System.Web.UI.Control)	Determines whether the event for the server control is passed up the page's UI server control hierarchy.

 OnCellExported	Called after grid cell is exported to Microsoft Excel format.
 OnCellExporting	Called before grid cell is exported to Microsoft Excel format.
 OnDataBinding (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.DataBinding event.
 OnEndExport	Called after grid export is finished.
 OnHeaderCellExported	Called after header cell is exported to Microsoft Excel format.
 OnHeaderCellExporting	Called before header cell is exported to Microsoft Excel format.
 OnHeaderRowExported	Called after header row is exported to Microsoft Excel format.
 OnHeaderRowExporting	Called before header row is exported to Microsoft Excel format.
 OnInit (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Init event.
 OnInitializeColumn	Occurs when a grid column is initialized.
 OnInitializeRow	Called when a row is initialized.
 OnLoad (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Load event.
 OnPreRender (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.PreRender event.
 OnRowExported	Called after grid row is exported to Microsoft Excel format.
 OnRowExporting	Called before grid row is exported to Microsoft Excel format.
 OnSummaryCellExported	Called after summary cell is exported to Microsoft Excel format.
 OnSummaryCellExporting	Called before summary cell is exported to Microsoft Excel format.
 OnSummaryRowExported	Called after summary row is exported to Microsoft Excel format.
 OnSummaryRowExporting	Called before summary row is exported to Microsoft Excel format.
 OnUnload (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Unload event. Server controls should perform any final cleanup, such as closing files, closing database connections, and discarding objects, during this stage of the server control lifecycle.
 RaiseBubbleEvent (Inherited from System.Web.UI.Control)	Assigns any sources of the event and its information to the control's parent.
 RemoveBadNameChars	Removes characters that may be inappropriate for Excel.
 RemovedControl (Inherited from System.Web.UI.Control)	Called after a control is removed from the System.Web.UI.Control.Controls collection of another control.
 Render	Overridden. Render this control to the output parameter specified.
 RenderChildren (Inherited from System.Web.UI.Control)	Outputs the content of a server control's children to a provided System.Web.UI.HtmlTextWriter object, which writes the content to be rendered on the client.

 SaveViewState (Inherited from System.Web.UI.Control)	Saves any server control view-state changes that have occurred since the time the page was posted back to the server.
 TrackViewState (Inherited from System.Web.UI.Control)	Causes tracking of view-state changes to the server control so they can be stored in the server control's System.Web.UI.StateBag object. This object is accessible through the System.Web.UI.Control.ViewState property.

See Also

[UltraWebGridExcelExporter Class](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

IGenerateGridExportEvents Interface

Interface describing the events developers may subscribe themselves to from the UltraWebGridExcelExporter. For a list of all members of this type, see [IGenerateGridExportEvents members](#).

Syntax

```
[Visual Basic]  
Public Interface IGenerateGridExportEvents
```

```
[C#]  
public interface IGenerateGridExportEvents
```

```
[JScript]  
public interface IGenerateGridExportEvents
```

See Also

[IGenerateGridExportEvents Members](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

IGenerateGridExportEvents Interface Members

[IGenerateGridExportEvents overview](#)

Public Events

 BeginExport	Occurs before the UltraWebGrid export starts.
 CellExported	Occurs after an UltraWebGrid cell is exported to Microsoft Excel format.
 CellExporting	Occurs before an UltraWebGrid cell is exported to Microsoft Excel format.
 EndExport	Occurs after the UltraWebGrid export is finished.
 HeaderCellExported	Occurs after header cell is exported to Microsoft Excel format.
 HeaderCellExporting	Occurs before header cell is exported to Microsoft Excel format.
 HeaderRowExported	Occurs after header row is exported to Microsoft Excel format.
 HeaderRowExporting	Occurs before header row is exported to Microsoft Excel format.
 InitializeColumn	Occurs when an UltraWebGrid column is initialized.
 InitializeRow	Occurs when each UltraWebGrid row is initialized.
 RowExported	Occurs after an UltraWebGrid row is exported to Microsoft Excel format.
 RowExporting	Occurs before an UltraWebGrid row is exported to Microsoft Excel format.
 SummaryCellExported	Occurs after summary cell is exported to Microsoft Excel format.
 SummaryCellExporting	Occurs before summary cell is exported to Microsoft Excel format.
 SummaryRowExported	Occurs after summary row is exported to Microsoft Excel format.
 SummaryRowExporting	Occurs before summary row is exported to Microsoft Excel format.

See Also

[IGenerateGridExportEvents Interface](#) | [Infragistics.WebUI.UltraWebGrid.ExcelExport Namespace](#)

Infragistics.WebUI.WebCombo Namespace

[Inheritance Hierarchy](#)

Classes

Class	Description
ClientSideEvents	
ColumnsListEditor	General-purpose implementation of an Editor for choosing a column name from the bound data source.
DropDownLayout	The properties of the DropDownLayout object control the look and behavior of the drop-down area of the control. Most of these properties are reflected directly down to the WebGrid control that is created as a child control of the WebCombo.
ExpandEffects	This object encapsulates the Internet Explorer Transitions functionality that WebCombo exposes.
SelectedRowChangedEventArgs	Class that contains information passed to SelectedRowChangedEvent handlers.
WebCombo	<p>The WebCombo control is a multi-column, editable drop-down selection control. It allows users to choose values from a detailed list of items that is shown when the arrow in the control is clicked.</p> <p>The drop-down area of WebCombo is based on the WebGrid control and supports many of the same features. Column and row sizing, sorting and row selectors are all supported within the drop-down. Because the WebCombo is intended for the selection of data values, editing within the drop-down is not supported.</p> <p>If a field can contain values beyond those in the list, the WebCombo can be set to editable so that values can be typed directly into the control.</p> <p>The WebCombo can be used as a stand-alone control anywhere on a web page that you would use a combobox control. WebCombo can also be used from within WebGrid to display ValueLists inside of grid cells.</p>

Interfaces

Interface	Description
ICanDropDown	

Enumerations

Enumeration	Description
ExpandEffectType	
TextAlignment	Enumeration for control of CheckBoxes at a particular level.
TypeAhead	Enumeration for typeahead behavior

Delegates

Delegate	Description
----------	-------------

InitializeFooterEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.InitializeFooter event, which is generated when footers need to be configured during data binding.
InitializeLayoutEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.InitializeLayout event, which is generated when the grid's layout needs to be initialized during data binding.
InitializeRowEventHandler	Delegate that handles the Infragistics.WebUI.UltraWebGrid.UltraWebGrid.InitializeRow event, which is generated when a row needs to be initialized during data binding.
SelectedRowChangedEventHandler	SelectedRowChangedEventHandler Delegate

See Also

[Infragistics.WebUI.WebCombo.v3.1 Assembly](#)

ClientSideEvents Class

For a list of all members of this type, see [ClientSideEvents members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.WebCombo.ClientSideEvents

Syntax

```
[Visual Basic]
Public Class ClientSideEvents
    Implements IStateManager
```

```
[C#]
public class ClientSideEvents : IStateManager
```

```
[JScript]
public class ClientSideEvents implements IStateManager
```


See Also

[ClientSideEvents Members](#) | [Infragistics.WebUI.WebCombo Namespace](#)


ClientSideEvents Class Members

[ClientSideEvents overview](#)


Public Constructors

 ClientSideEvents Constructor	The default public constructor for the ClientSideEvents object. This object is created automatically by the WebCombo control.
---	---

Public Properties

 AfterCloseUp	Returns or sets a string of text that specifies the name of the JavaScript function to be called immediately after the dropdown portion of the control has been closed up.
 AfterDropDown	Returns or sets a string of text that specifies the name of the JavaScript function to be called immediately after the dropdown portion of the control has been displayed.
 AfterSelectChange	Returns or sets a string of text that specifies the name of the JavaScript function to be called immediately after the selected row in the dropdown portion of the control has changed.
 BeforeCloseUp	Returns or sets a string of text that specifies the name of the JavaScript function to be called prior to the dropdown portion of the control closing up.
 BeforeDropDown	Returns or sets a string of text that specifies the name of the JavaScript function to be called prior to the dropdown portion of the control being displayed.
 BeforeSelectChange	Returns or sets a string of text that specifies the name of the JavaScript function to be called prior to the selected row in the dropdown portion of the control being changed.
 EditKeyDown	Returns or sets a string of text that specifies the name of the JavaScript function to be called when a key has been pressed down within the edit portion of the control.
 EditKeyUp	Returns or sets a string of text that specifies the name of the JavaScript function to be called when a key has been pressed down within the edit portion of the control.
 InitializeCombo	Returns or sets a string of text that specifies the name of the JavaScript function to be called when the Combo has been initialized and is ready for display.

Public Methods

 ToString	Overridden.
---	-------------

Protected Properties

 ViewState	
--	--

See Also

[ClientSideEvents Class](#) | [Infragistics.WebUI.WebCombo Namespace](#)

ColumnsListEditor Class

General-purpose implementation of an Editor for choosing a column name from the bound data source.
For a list of all members of this type, see [ColumnsListEditor members](#).

Inheritance Hierarchy

[System.Object](#)
[System.Drawing.Design.UITypeEditor](#)
Infragistics.WebUI.WebCombo.ColumnsListEditor

Syntax

```
[Visual Basic]  
Public Class ColumnsListEditor  
    Inherits UITypeEditor
```

```
[C#]  
public class ColumnsListEditor : UITypeEditor
```

```
[JScript]  
public class ColumnsListEditor extends UITypeEditor
```

See Also

[ColumnsListEditor Members](#) | [Infragistics.WebUI.WebCombo Namespace](#)





ColumnsListEditor Class Members

[ColumnsListEditor overview](#)

Public Constructors

 ColumnsListEditor Constructor	
--	--

Public Methods

 EditValue	Overloaded. Overridden.
 GetEditStyle	Overloaded. Overridden.
 GetPaintValueSupported	Overloaded. Overridden.
 PaintValue (Inherited from System.Drawing.Design.UITypeEditor)	Overloaded.

See Also

[ColumnsListEditor Class](#) | [Infragistics.WebUI.WebCombo Namespace](#)

DropDownLayout Class

The properties of the DropDownLayout object control the look and behavior of the drop-down area of the control. Most of these properties are reflected directly down to the WebGrid control that is created as a child control of the WebCombo.

For a list of all members of this type, see [DropDownLayout members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.WebCombo.DropDownLayout

Syntax

```
[Visual Basic]  
Public Class DropDownLayout
```

```
[C#]  
public class DropDownLayout
```

```
[JScript]  
public class DropDownLayout
```









See Also









[DropDownLayout Members](#) | [Infragistics.WebUI.WebCombo Namespace](#)

DropDownLayout Class Members


[DropDownLayout overview](#)

Public Properties

 AllowColSizing	Determines whether the user is allowed to change the width of columns in the dropdown.
 AllowRowSizing	Specifies whether and how the user can change the height of the rows in the dropdown.
 AllowSorting	Specifies whether the user can sort columns.
 AutoGenerateColumns	Specifies whether the combo should generate any columns automatically when data binding occurs.
 BorderCollapse	
 CellPadding	The amount of cell padding to use when rendering the dropdown.
 CellSpacing	The amount of cell spacing to use when rendering the grid.
 ColFootersVisible	Determines whether to show column footers for the dropdown.
 ColHeadersVisible	Determines whether to show column headers in the dropdown.
 ColWidthDefault	The default width of columns in the dropdown, in Units.
 DropDownHeight	Returns or sets a System.Web.UI.WebControls.Unit value that determines the height of the dropdown.
 DropDownWidth	Returns or sets a System.Web.UI.WebControls.Unit value that determines how wide the dropdown will be when the user clicks the dropdown button.
 FooterStyle	The default style that will be applied to column footers.
 FrameStyle	An Infragistics.WebUI.UltraWebGrid.UltraWebStyle object that contains the style properties for the control's frame.
 GridLines	Specifies which cell borders should be shown. The default is UltraGridLines.Both.
 HeaderClickAction	Specifies what action to take when column headers are clicked.
 HeaderStyle	The default style that will be applied to headers. This is the default value that may be overridden by individual columns.
 ImageUrls	Object holder for images used in the dropdown. Use the properties of this object to customize the various images that are displayed within the dropdown area of WebCombo.
 JavaScriptFileName	The base name and path for the JavaScript files that are used for client side behavior. The root name of the file, (ig_WebGrid) is used as the base for determining the other files that are contain related functionality. If this root file name is changed, then all other file names must be changed as well.
 RowAlternateStyle	The default style that will be applied to alternate (even numbered) rows in the dropdown.

 RowHeightDefault	The default height that will be applied to all rows in the dropdown.
 RowSelectors	Specifies whether row selectors will be displayed in the dropdown.
 RowSelectorStyle	The default style that will be applied to row selectors in the dropdown.
 RowSizing	Specifies whether and how the user can change the height of the rows in the dropdown. The default is AllowSizing.Fixed.
 RowStyle	The default style that will be applied to rows.
 SelectedRowStyle	The style that will be applied to the selected row in the dropdown.
 StationaryMargins	
 TableLayout	The type of table layout in effect for the dropdown. If this value is set to Auto, the heights and widths of rows and columns are interpreted by the browser as a recommendation. The Browser will still make adjustments as needed to make as much information as possible visible. If this property value is set to Fixed, the Browser will treat row and column heights and widths literally and force them to the indicated values. This option is only effective on up-level browsers.

Public Methods

 ToString	Overridden.
---	-------------

See Also

[DropDownLayout Class](#) | [Infragistics.WebUI.WebCombo Namespace](#)

ExpandEffects Class

This object encapsulates the Internet Explorer Transitions functionality that WebCombo exposes. For a list of all members of this type, see [ExpandEffects members](#).

Inheritance Hierarchy

[System.Object](#)

Infragistics.WebUI.WebCombo.ExpandEffects

Syntax

```
[Visual Basic]
Public Class ExpandEffects
    Implements IStateManager
```

```
[C#]
public class ExpandEffects : IStateManager
```

```
[JScript]
public class ExpandEffects implements IStateManager
```

See Also

[ExpandEffects Members](#) | [Infragistics.WebUI.WebCombo Namespace](#)






ExpandEffects Class Members

[ExpandEffects overview](#)


Public Constructors

 ExpandEffects Constructor	
--	--

Public Properties

 Duration	Specifies the amount of time elapsed in milliseconds between the beginning and the end of a transition effect.
 Opacity	Specifies the amount of transparency applied to the expandable object after the expand effect is complete.
 ShadowColor	Specifies the color of the drop shadow that will appear under the expandable object.
 ShadowWidth	Specifies the width of the drop shadow that will appear under the expandable object.
 Type	

Public Methods

 ToString	Overridden. Returns a string representation of the ExpandEffects object.
---	--

Protected Properties

 ViewState	
--	--

See Also

[ExpandEffects Class](#) | [Infragistics.WebUI.WebCombo Namespace](#)

SelectedRowChangedEventArgs Class

Class that contains information passed to SelectedRowChangedEventArgs handlers.

For a list of all members of this type, see [SelectedRowChangedEventArgs members](#).

Inheritance Hierarchy

[System.Object](#)

[System.EventArgs](#)

Infragistics.WebUI.WebCombo.SelectedRowChangedEventArgs

Syntax

```
[Visual Basic]
Public Class SelectedRowChangedEventArgs
    Inherits EventArgs
```

```
[C#]
public class SelectedRowChangedEventArgs : EventArgs
```

```
[JScript]
public class SelectedRowChangedEventArgs extends EventArgs
```

See Also

[SelectedRowChangedEventArgs Members](#) | [Infragistics.WebUI.WebCombo Namespace](#)

SelectedRowChangedEventArgs Class Members

[SelectedRowChangedEventArgs overview](#)

See Also

[SelectedRowChangedEventArgs Class](#) | [Infragistics.WebUI.WebCombo Namespace](#)

WebCombo Class

The WebCombo control is a multi-column, editable drop-down selection control. It allows users to choose values from a detailed list of items that is shown when the arrow in the control is clicked.

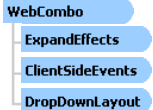
The drop-down area of WebCombo is based on the WebGrid control and supports many of the same features. Column and row sizing, sorting and row selectors are all supported within the drop-down. Because the WebCombo is intended for the selection of data values, editing within the drop-down is not supported.

If a field can contain values beyond those in the list, the WebCombo can be set to editable so that values can be typed directly into the control.

The WebCombo can be used a stand-alone control anywhere on a web page that you would use a combobox control. WebCombo can also be used from within WebGrid to display ValueLists inside of grid cells.

For a list of all members of this type, see [WebCombo members](#).

Object Model



Inheritance Hierarchy

- [System.Object](#)
- [System.Web.UI.Control](#)
- [System.Web.UI.WebControls.WebControl](#)
- Infragistics.WebUI.WebCombo.WebCombo**

Syntax

```
[Visual Basic]
Public Class WebCombo
    Inherits WebControl
    Implements
        IComponent, IDisposable, IParseAccesser, IDataBindingsAccessor, IAttributeAccessor, IPostBackDataHandler, IPostBackEventHandler, INamingContainer, IUltraLicensedComponent, IProvidesEmbeddableEditor, IGetClientSideEvents, ICanDropDown, IProvideDesignTimeHtml, ISupportPresetSerialization
```

```
[C#]
public class WebCombo : WebControl,
    IComponent, IDisposable, IParseAccesser, IDataBindingsAccessor, IAttributeAccessor, IPostBackDataHandler, IPostBackEventHandler, INamingContainer, IUltraLicensedComponent, IProvidesEmbeddableEditor, IGetClientSideEvents, ICanDropDown, IProvideDesignTimeHtml, ISupportPresetSerialization
```

```
[JScript]
public class WebCombo extends WebControl implements
    IComponent, IDisposable, IParseAccesser, IDataBindingsAccessor, IAttributeAccessor, IPostBackDataHandler, IPostBackEventHandler, INamingContainer, IUltraLicensedComponent, IProvidesEmbeddableEditor, IGetClientSideEvents, ICanDropDown, IProvideDesignTimeHtml, ISupportPresetSerialization
```


See Also

- [WebCombo Members](#) | [Infragistics.WebUI.WebCombo Namespace](#)















WebCombo Class Members





















[WebCombo overview](#)

Public Constructors







 WebCombo Constructor	Constructor for the WebCombo control
---	--------------------------------------

Public Properties










 AccessKey (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the access key (underlined letter) that allows you to quickly navigate to the Web server control.
 Attributes (Inherited from System.Web.UI.WebControls.WebControl)	Gets the collection of arbitrary attributes (for rendering only) that do not correspond to properties on the control.
 BackColor (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the background color of the Web server control.
 BindingContainer (Inherited from System.Web.UI.Control)	
 BorderColor (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the border color of the Web control.
 BorderStyle (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the border style of the Web server control.
 BorderWidth (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the border width of the Web server control.
 ClientID (Inherited from System.Web.UI.Control)	Gets the server control identifier generated by ASP.NET.
 ClientSideEvents	The ClientSideEvents object maintains the names of JavaScript functions that are to be called on the client side in response to various events that can be processed without the need for a server post-back.
 Columns	Returns a reference to a Infragistics.WebUI.UltraWebGrid.ColumnsCollection collection that contains a collection of Infragistics.WebUI.UltraWebGrid.UltraGridColumn objects.
 ComboTypeAhead	Determines what typeahead behavior the control will abide by
 CompactRendering	Returns or sets a boolean value that determines whether or not the HTML is rendered in a compact format that is less readable but which optimizes page download time. When this property is set to true, extra spaces, tabs and line feed characters are not used in the rendered html.
 Controls (Inherited from System.Web.UI.Control)	Gets a System.Web.UI.ControlCollection object that represents the child controls for a specified server control in the UI hierarchy.
 ControlStyle (Inherited from System.Web.UI.WebControls.WebControl)	Gets the style of the Web server control. This property is used primarily by control developers.







 ControlStyleCreated (Inherited from System.Web.UI.WebControls.WebControl)	Gets a value indicating whether a System.Web.UI.WebControls.Style object has been created for the System.Web.UI.WebControls.WebControl.ControlStyle property. This property is primarily used by control developers.
 CssClass (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the Cascading Style Sheet (CSS) class rendered by the Web server control on the client.
 DataMember	DataMember is used as the starting table in the DataSet hierarchy for binding.
 DataSource	Gets or sets the data source that populates the grid with data.
 DataTextField	Specifies the column name from the dropdown that will be used to display the text content of the top portion of the WebCombo.
 DataValue	The text content string of the top portion of the WebCombo.
 DataValueField	Specifies the column name from the dropdown that will be used as the source for the DataValue of the WebCombo control.
 DisplayValue	The text content string of the top portion of the WebCombo.
 DropDownLayout	The UltraGridLayout object used to control the look and behavior of the dropdown portion of the WebCombo control.
 DropImage1	The Image used to represent the non-dropped state of the control
 DropImage2	The image used to indicate that the WebCombo drop down is being displayed.
 DropImageXP1	The Image used to represent the non-dropped state of the control in the XP operating system environment.
 DropImageXP2	The image used to indicate that the WebCombo drop down is being displayed in the XP operating system environment.
 Editable	Determines if the WebCombo control allows values to be entered by the user.
 Enabled (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets a value indicating whether the Web server control is enabled.
 EnableViewState (Inherited from System.Web.UI.Control)	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client.
 ExpandEffects	Returns the ExpandEffects object that determines the type of filters and transitions that will be used when displaying sub menus under IE 5 and above.
 Font (Inherited from System.Web.UI.WebControls.WebControl)	Gets the font properties associated with the Web server control.
 ForeColor (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the foreground color (typically the color of the text) of the Web server control.
 Height (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the height of the Web server control.

 HideDropDowns	Returns or sets a boolean value that determines whether or not Select dropdown controls will be automatically hidden when submenus are being displayed. When this property is set to true, all select controls in the page document are hidden until the popup menus are dismissed.
 ID (Inherited from System.Web.UI.Control)	Gets or sets the programmatic identifier assigned to the server control.
 JavaScriptFileName	The path and name for the JavaScript file used for client-side dropdown behavior.
 JavaScriptFileNameCommon	The relative path to the common javascript file.
 NamingContainer (Inherited from System.Web.UI.Control)	Gets a reference to the server control's naming container, which creates a unique namespace for differentiating between server controls with the same System.Web.UI.Control.ID property value.
 Page (Inherited from System.Web.UI.Control)	Gets a reference to the System.Web.UI.Page instance that contains the server control.
 Parent (Inherited from System.Web.UI.Control)	Gets a reference to the server control's parent control in the page control hierarchy.
 Rows	Returns a reference to a Infragistics.WebUI.UltraWebGrid.RowsCollection collection that contains a collection of Infragistics.WebUI.UltraWebGrid.UltraGridRow objects.
 SelBackColor	Determines the Selected Background Color for the WebCombo when the input focus is in the top area of the control.
 SelectedCell	Returns a reference to the currently selected Cell object. The selected cell is determined by a combination of which row is currently selected, and which column within the row has been designated as the data column. The data column is designated by setting either the DataTextField property or the first cell in the row. The properties are examined in the order specified (Text then the first cell) to make the determination of which cell within the row is the Selected Cell.
 SelectedIndex	Returns or sets the index of the currently selected Row in the drop-down.
 SelectedRow	Returns a reference to or sets the currently selected Row object.
 SelForeColor	Determines the Selected Foreground Color for the WebCombo when the input focus is in the top area of the control.
 Site (Inherited from System.Web.UI.Control)	Gets information about the Web site to which the server control belongs.
 Style (Inherited from System.Web.UI.WebControls.WebControl)	Gets a collection of text attributes that will be rendered as a style attribute on the outer tag of the Web server control.
 TabIndex (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the tab index of the Web server control.











 TemplateSourceDirectory (Inherited from System.Web.UI.Control)	Gets the virtual directory of the System.Web.UI.Page or System.Web.UI.UserControl that contains the current server control.
 ToolTip (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the text displayed when the mouse pointer hovers over the Web server control.
 UniqueID (Inherited from System.Web.UI.Control)	Gets the unique, hierarchically-qualified identifier for the server control.
 Version	
 Visible (Inherited from System.Web.UI.Control)	Gets or sets a value that indicates whether a server control is rendered as UI on the page.
 Width (Inherited from System.Web.UI.WebControls.WebControl)	Gets or sets the width of the Web server control.

Public Methods








 ApplyStyle (Inherited from System.Web.UI.WebControls.WebControl)	Copies any nonblank elements of the specified style to the Web control, overwriting any existing style elements of the control. This method is primarily used by control developers.
 CopyBaseAttributes (Inherited from System.Web.UI.WebControls.WebControl)	Copies the properties not encapsulated by the System.Web.UI.WebControls.WebControl.Style object from the specified Web server control to the Web server control that this method is called from. This method is used primarily by control developers.
 DataBind	Overridden. The method used to invoke databinding for the WebCombo control
 Dispose	
 FindByText	Locates the first Cell in the DataTextColumn of the WebCombo that matches the passed in value. The Row containing the returned cell is available from the cell. Row property.
 FindByValue	Locates the first Cell in the DataValueColumn of the WebCombo that matches the passed in value. The Row containing the returned cell is available from the cell. Row property.
 FindControl (Inherited from System.Web.UI.Control)	Overloaded.
 HasControls (Inherited from System.Web.UI.Control)	Determines if the server control contains any child controls.
 LoadPostData	Not for use; implemented exclusively for the control and the .NET framework.
 LoadPreset	Overloaded.
 MergeStyle (Inherited from System.Web.UI.WebControls.WebControl)	Copies any nonblank elements of the specified style to the Web control, but will not overwrite any existing style elements of the control. This method is used primarily by control developers.
 OnSelectedRowChanged	Raises the NodeChanged event.
 RaisePostBackEvent	Used by control authors. Overridden method of WebCombo that is used to handle event postbacks from the client.
 RaisePostBackDataChangedEvent	Not for use; implemented exclusively for the control and the .NET framework.



 RenderBeginTag (Inherited from System.Web.UI.WebControls.WebControl)	Renders the HTML opening tag of the control into the specified writer. This method is used primarily by control developers.
 RenderControl (Inherited from System.Web.UI.Control)	Outputs server control content to a provided System.Web.UI.HtmlTextWriter object and stores tracing information about the control if tracing is enabled.
 RenderEndTag (Inherited from System.Web.UI.WebControls.WebControl)	Renders the HTML closing tag of the control into the specified writer. This method is used primarily by control developers.
 ResolveUrl (Inherited from System.Web.UI.Control)	Converts a URL into one that is usable on the requesting client.
 SavePreset	Overloaded.
 SetRenderMethodDelegate (Inherited from System.Web.UI.Control)	

Public Events





 DataBinding (Inherited from System.Web.UI.Control)	
 Disposed (Inherited from System.Web.UI.Control)	
 Init (Inherited from System.Web.UI.Control)	
 InitializeFooter	Manages delegates of the type Infragistics.WebUI.UltraWebGrid.InitializeRowEventHandler .
 InitializeLayout	Manages delegates of the type Infragistics.WebUI.UltraWebGrid.InitializeRowEventHandler .
 InitializeRow	Manages delegates of the type Infragistics.WebUI.UltraWebGrid.InitializeRowEventHandler .
 Load (Inherited from System.Web.UI.Control)	
 PreRender (Inherited from System.Web.UI.Control)	
 SelectedRowChanged	Fired when the selected Row in the dropdown has been changed on the client
 Unload (Inherited from System.Web.UI.Control)	





Protected Properties

 ChildControlsCreated (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control's child controls have been created.
 Context (Inherited from System.Web.UI.Control)	Gets the System.Web.HttpContext object associated with the server control for the current Web request.
 Events (Inherited from System.Web.UI.Control)	Gets a list of event handler delegates for the control. This property is read-only.
 HasChildViewState (Inherited from System.Web.UI.Control)	Gets a value indicating whether the current server control's child controls have any saved view-state settings.
 IsTrackingViewState (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the server control is saving changes to its view state.
 TagKey (Inherited from System.Web.UI.WebControls.WebControl)	Gets the System.Web.UI.HtmlTextWriterTag value that corresponds to this Web server control. This property is used primarily by control developers.
 TagName (Inherited from System.Web.UI.WebControls.WebControl)	Gets the name of the control tag. This property is used primarily by control developers.

 ViewState (Inherited from System.Web.UI.Control)	Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page.
 ViewStateIgnoresCase (Inherited from System.Web.UI.Control)	Gets a value that indicates whether the System.Web.UI.StateBag object is case-insensitive.

Protected Methods

 AddAttributesToRender	Overridden. Not to use.
 AddedControl (Inherited from System.Web.UI.Control)	Called after a control is added to the System.Web.UI.Control.Controls collection of another control.
 AddParsedSubObject (Inherited from System.Web.UI.Control)	Notifies the server control that an element, either XML or HTML, was parsed, and adds the element to the server control's System.Web.UI.ControlCollection object.
 BuildProfileTree (Inherited from System.Web.UI.Control)	
 ClearChildViewState (Inherited from System.Web.UI.Control)	Deletes the view-state information for all the server control's child controls.
 CreateChildControls	Overridden. Not to use. Overridden by the WebCombo control to create the child control structure.
 CreateControlCollection (Inherited from System.Web.UI.Control)	Creates a new System.Web.UI.ControlCollection object to hold the child controls (both literal and server) of the server control.
 CreateControlStyle	Overridden. Creates the Style object for the control.
 EnsureChildControls (Inherited from System.Web.UI.Control)	Determines whether the server control contains child controls. If it does not, it creates child controls.
 IsLiteralContent (Inherited from System.Web.UI.Control)	Determines if the server control holds only literal content.
 LoadViewState	Overridden.
 MapPathSecure (Inherited from System.Web.UI.Control)	Retrieves the physical path that a virtual path, either absolute or relative, maps to.
 OnBubbleEvent (Inherited from System.Web.UI.Control)	Determines whether the event for the server control is passed up the page's UI server control hierarchy.
 OnDataBinding (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.DataBinding event.
 OnInit (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Init event.
 OnLoad (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Load event.
 OnPreRender	Overridden. Prepare the renderer and emit JavaScript
 OnUnload (Inherited from System.Web.UI.Control)	Raises the System.Web.UI.Control.Unload event. Server controls should perform any final cleanup, such as closing files, closing database connections, and discarding objects, during this stage of the server control lifecycle.
 RaiseBubbleEvent (Inherited from System.Web.UI.Control)	Assigns any sources of the event and its information to the control's parent.
 RemovedControl (Inherited from System.Web.UI.Control)	Called after a control is removed from the System.Web.UI.Control.Controls collection of another control.
 Render	Overridden. Render this control to the output parameter specified.

 RenderChildren (Inherited from System.Web.UI.Control)	Outputs the content of a server control's children to a provided System.Web.UI.HtmlTextWriter object, which writes the content to be rendered on the client.
 RenderContents (Inherited from System.Web.UI.WebControls.WebControl)	Renders the contents of the control into the specified writer. This method is used primarily by control developers.
 SaveViewState	Overridden.
 TrackViewState	Overridden.

See Also

[WebCombo Class](#) | [Infragistics.WebUI.WebCombo Namespace](#)

ICanDropDown Interface

For a list of all members of this type, see [ICanDropDown members](#).

Syntax

```
[Visual Basic]  
Public Interface ICanDropDown
```

```
[C#]  
public interface ICanDropDown
```

```
[JScript]  
public interface ICanDropDown
```

See Also

[ICanDropDown Members](#) | [Infragistics.WebUI.WebCombo Namespace](#)

ICanDropDown Interface Members

[ICanDropDown overview](#)

Public Properties

 DropDownVisible	
--	--

See Also

[ICanDropDown Interface](#) | [Infragistics.WebUI.WebCombo Namespace](#)