

# WebGrid Enterprise 7

---

This white paper explains the new features, enhancements, and breaking changes introduced in WebGrid Enterprise 7.

## Contents

Overview.....	4
Data editing with new Batch Update mode.....	5
Introducing SmartBatchUpdate™.....	5
Pending Changes Concept .....	6
Rich User Interfaces.....	8
Streamlined Editing Process.....	10
Built-in Changes Management.....	12
Hierarchical Tables Support .....	14
Automatic Changes Preservation .....	16
Updating to Physical Database.....	16
Understanding Batch Update Processes.....	17
Various Datasource Support .....	20
New Identity Insert .....	22
Localization .....	26
Compatibility with Other Features.....	28
Client-side Programmability.....	29
New classes .....	29
New methods .....	30
New client side events .....	31
Server-side Programmability.....	32
Client-side data binding.....	36
Overview .....	36
What is client-side binding? .....	36
Innovative VirtualRendering™ .....	37
Benefits .....	38

Performance Benchmark and Comparison .....	39
Supported Features and Limitations.....	41
The Differences with Server-side Binding .....	44
Elegant Client Binding Architecture.....	47
Client Data Object Framework .....	47
Rich Client-side Data Processing.....	49
Data Loading Mode.....	51
Transaction Operations (Insert, Update and Delete) .....	53
Batch Update Support.....	55
Service Events.....	56
Client Data Source and Data Service .....	57
Server-side data source.....	57
Web Service.....	58
Windows Communication Foundation (WCF) Service.....	58
ADO.NET Data Service (Astoria) .....	59
Client-side data source .....	59
Client Binding Consideration and Best Practices.....	60
Considerations.....	60
Best practice #1 – Using ServerDataSource with Paging enabled .....	60
Best practice #2 – Using WebService with PagedData loading mode .....	60
Best practice #3 – Using AdoDataService .....	60
Limitations.....	60
Client Binding Configuration with Component Designer.....	61
Client Binding API .....	62
New methods .....	62
New client side events .....	63
Other Enhancements.....	64
Improved user interface .....	64
New client side events for inline editing .....	65
Enhanced integration with WebCombo .....	65
Default Style Merging .....	66
Without Default Style Merging.....	66

With Default Style Merging.....	66
---------------------------------	----

## Overview

WebGrid Enterprise™ 7 is the next-generation data visualization component for ASP.NET that revolutionizes data editing experience and improves reliability by addressing performance bottlenecks with a sophisticated client binding implementation.

WebGrid Enterprise™ 7 is strongly focused on enterprise data application that takes advantage of new Web technologies such as “cloud” data service and offline-capable editing. This new version includes re-engineered architecture to fully support client data operation, which is the key answer to client-side binding while preserving the compatibilities with existing advanced features.

Intersoft’s ClientBinding™ is the technology behind WebGrid’s client binding implementation, which reduces data footprint by over 90 percent and improves overall response time by 10 times. See the [performance benchmark](#) for more information. In addition, client binding also introduces numerous benefits. Learn more about [client binding](#) and its [benefits](#).

One of the key features of ClientBinding™ is its high-level encapsulation that enables you to easily leverage existing investments of your data service infrastructure. ClientBinding™ enables you to connect to data service in an elegant way, without requiring you to write Javascript codes. See [Elegant Client Binding Concept](#) to learn how you can quickly getting started with client binding.

Use [Component Designer](#) which has been redesigned to help you easily configure client binding settings in WebGrid. With so many features provided by ClientBinding™ such as data source type, loading mode, transaction support and more, please refer to [client binding best practices](#) to help you find the best configuration for your client binding implementation.

WebGrid Enterprise™ 7 also features SmartBatchUpdate™, a new major innovation that takes editing experience to a new level. This new technology enables your end user to make multiple changes to the data in the client side, while at the same time maintaining the changes as they navigate the data around.

Embracing [elegant pending changes architecture](#), WebGrid submits all changes to server-side in a single request – making data update fast and efficient. Please see [SmartBatchUpdate™ overview](#) to learn the fundamental concept of this new feature.

WebGrid 7’s batch update also supports advanced data transaction operations such as cascading inserts, [hierarchical tables](#), [automatic object updates](#), [identity insert handling](#) and more. The user interface has also been improved to provide users with intuitive way to review and manage the changes. Learn more about user [interface improvements](#) such as call out notification, changes status and more.

The new technologies in WebGrid 7 are also designed to work in concert with each other, making it the most advanced data visualization and management component for dynamic, Web 2.0-enabled application.

Learn how to [combine client binding and batch update](#) along with inline editing to deliver powerful, “cloud”-ready and offline-capable Web application.

## Data editing with new Batch Update mode

### Introducing SmartBatchUpdate™

SmartBatchUpdate™ is a new feature in WebGrid Enterprise 7 which enables you to perform multiple edits across multiple tables in real-time without postback/callback. All pending changes will be submitted into server at once with a single AJAX callback, thus eliminates waiting time and improves data editing experience in overall.

When WebGrid is operating in batch update editing mode, you can make multiple edits – such as *adding new row*, *editing row*, and *deleting row* – in real-time without server contact. You can also make multiple edits across hierarchical child tables which are linked through valid referential integrity in the same consistent fashion.

SmartBatchUpdate™ also supports more advanced scenarios such as *cascading inserts* and *deletes* on hierarchical tables. This allows you to conveniently add new master record and its child records in a row – making data entry across hierarchical tables a snap. Furthermore, WebGrid stores the relation between each new record and intelligently perform new identity translation and mapping it to related child records during physical database updates – freeing developers from tedious and lengthy codes.

SmartBatchUpdate™ provides numerous benefits for developers and end users, such as:

- **Improves data editing experience.**  
Unlike traditional data editing, WebGrid doesn't trigger page postback/callback when a row edit occurred. By eliminating the wait time for each row edits, end users can perform data editing faster and more efficient than ever.
- **Reduces server workload.**  
SmartBatchUpdate™ saves the changes of each modified row entirely in client-side which allows WebGrid to efficiently manage all pending changes without have to go back-and-forth to server. When end-users are ready to commit changes, WebGrid will send a single AJAX callback to submit all changes to be processed at the server. That translates to more efficient resource utilization and minimized server workload.
- **Prevents data entry errors.**  
SmartBatchUpdate™ includes powerful runtime features to help you make correct changes on data and prevent erroneous changes which you want to avoid at all costs. The *Undo Changes* feature lets end-users to undo a row changes before it is sent to server for processing. The *Review Changes* feature enables end-users to review all pending changes in an intuitive dialog box interface, where end-users can make correction on specific changes across multiple tables, or undo the changes.

## Pending Changes Concept

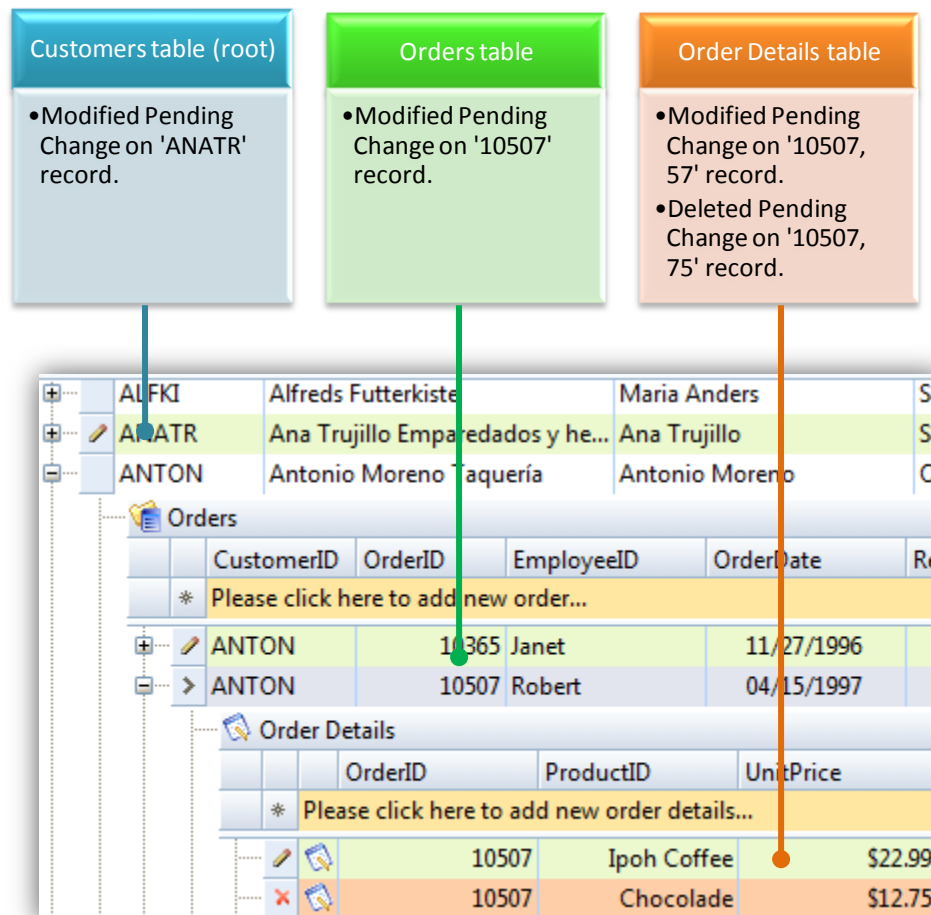
SmartBatchUpdate™ employs pending changes concept to provide solid functionality and architecture for its batch update features. With the concept, every row that has changed since its first load will result in a pending change.

A record row can contain only one pending change at a time, which is one of four modes below:

- **Unmodified.** When a change is undo'ed, the record will be set back to Unmodified.
- **Added.** Newly added row will be marked as Added pending change.
- **Modified.** Edited row will be marked as Modified pending change.
- **Deleted.** Deleted row will be marked as Deleted pending change.

Each row's pending change is stored based on its corresponding table. This means that each table contains one or more pending changes which are associated with each logical row. As a result, SmartBatchUpdate™ provides solid and consistent object models and interfaces which enable pending changes to be consumed in multiple tables (hierarchical) configuration. For more information about batch update support for Hierarchical feature, please read [Hierarchical Tables Support](#)

The following illustration describes the pending changes concept in a hierarchical Grid.



Pending changes are stored locally on client-side and can be accessed programmatically through [client-side API](#).

Furthermore, pending changes are [automatically restored and synchronized](#) with the current view whenever the Grid performed a FlyPostBack action – such as sorting, filtering, grouping, etc – or page full postback. This provides users with greater experience to interacting with information while maintaining current changes simultaneously.

SmartBatchUpdate™ also includes built-in pending changes management, such as ability to undo changes, accept changes, as well as review changes. For more information about pending changes management, please see [Built-in Changes Management](#).

It's important to note that WebGrid requires your tables to have at least one unique key field, which is assigned to *DataKeyField* property of each *WebGridTable* instance. Multiple key fields' scenario is also supported.

When *Accept Changes* command is invoked by users, WebGrid submits all pending changes to server in a single FlyPostBack (AJAX) callback. Upon receiving batch update request, WebGrid will automatically apply all pending changes to the data source in batch. Depending on your data source type, WebGrid will decide whether it should automatically apply the changes to physical database. To learn more about physical database updates, please visit [Updating to Physical Database](#).

With solid and extensible pending changes architecture provided by SmartBatchUpdate™, WebGrid Enterprise 7 delivers reliable client-side editing solution for your Web application, where data lost is not an option.

## Rich User Interfaces

In addition to solid editing architecture and powerful runtime features, SmartBatchUpdate™ also provides your end-users with rich visual elements to easily determine added, modified and deleted rows.

The following image shows a WebGrid in hierarchical tables with several pending changes.

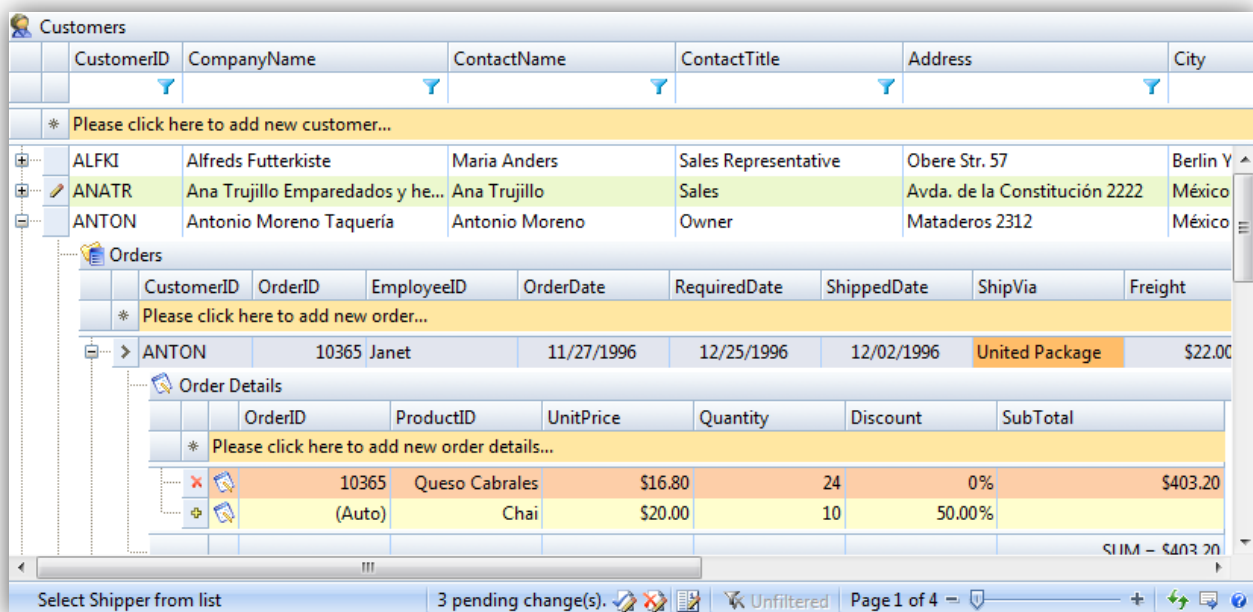
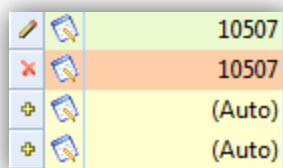


Figure 1. WebGrid provides rich visual hints and indicators for pending changes.

SmartBatchUpdate™ includes additional visual elements and indicators, such as explained below:

- **Changes indicator in row header.**



The visual indicator in row header allows users to quickly distinguish the changes status of each row. Added row is marked with icon; modified row is marked with icon; while deleted row is marked with icon.

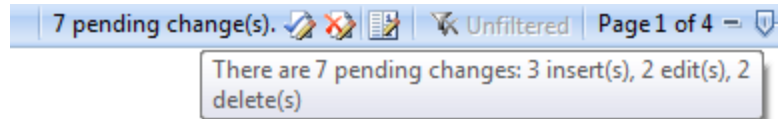
- **Visual row style.**

Different row change status will have different row style to let users recognize the status of each change instantly. By default, added rows are marked with light yellow, modified rows with light green, and deleted rows with light red. These visual styles can be customized through *AddedRowStyle*, *ModifiedRowStyle* and *DeletedRowStyle* respectively.



- **Pending changes status and related commands in status bar.**

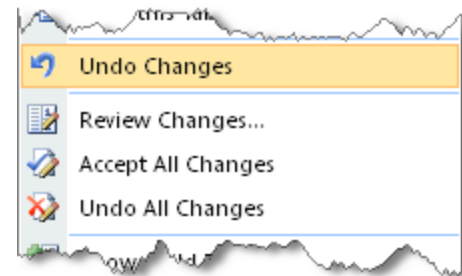
Whenever users make changes to data, add or remove a row, WebGrid will automatically maintain the pending changes status by updating the related user interface elements in the status bar area. The pending changes status makes it easy for users to understand the current



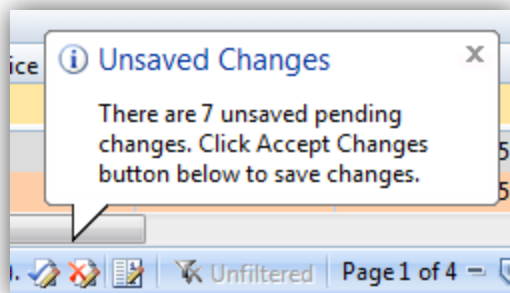
changes state, as well as to perform an action that related to the current changes such as accepting all changes.

- **Integration with context menu.**

SmartBatchUpdates™ naturally integrates into existing context menu interface. WebGrid will display various pending changes related command depending on the row state and the settings provided in *BatchUpdateSettings* object.



- **Callout notification on lost focus.**

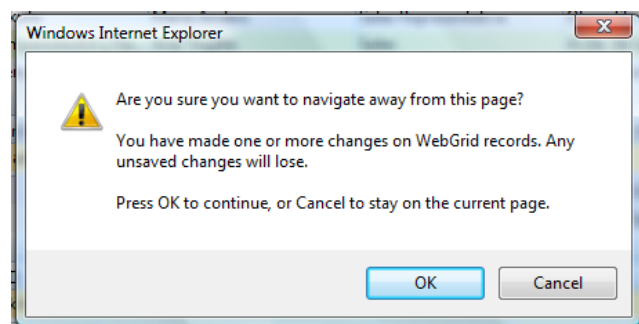


Users tend to forget to save changes when they are doing multiple tasks at the same time. With the eye-catching Vista-style notification in WebGrid 7, that is nothing to worry about.

The Vista-style notification will appear automatically when WebGrid lose its focus and pending changes existed.

- **Prompt on page navigation**

Better yet – users will be prompted when they are about to leave the page while pending changes existed. This feature is especially useful to avoid changes lost due to accidental hyperlink clicks or browser close.



## Streamlined Editing Process

SmartBatchUpdate™ incorporates streamlined editing behaviors to make it even easier, faster and more convenient for users to work with data.

When the batch update feature is enabled, the following behaviors will be automatically enabled:

- **Pending changes merging.**

A record can contain only one mode of changes. Therefore, if you perform several changes on the same mode, the pending changes will merge automatically. If the change mode is different, it will remove the previous changes and set the latest change mode as active.

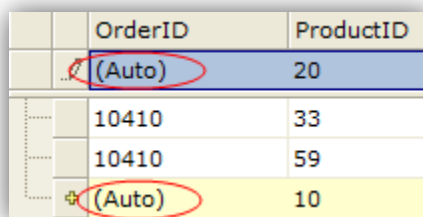
For instances, consider the following scenarios:

- User edited ContactName on record A. Next, CustomerName and Country are changed. The latter changes will be *merged with first changes*, resulting in single pending change that contains three fields edit.
- User edited a record then later deletes it. In such case, the previous changes will be undo'ed and the *delete pending change is added*.
- User added a record then later edits more fields. The record will *remain marked as added pending change*, while the latter edits will be merged.
- User added a record the later deletes it. In such case, the record will be *physically deleted from client view*. Its pending change will be removed as well.

- **Go to next row on last cell.**

When you pressed TAB key to edit data and past the last cell of the row, the edited row will be marked as pending changes. The active cell selection will also be set to the next row, which allows users to edit data faster.

- **Unique auto-increment fields will be set to (Auto).**



	OrderID	ProductID
	(Auto)	20
	10410	33
	10410	59
	(Auto)	10

WebGrid will automatically set auto-increment column – which is usually used as *DataKeyField* – as read-only to avoid unnecessary data update error. Furthermore, the auto-increment field will use a user-friendly text such as “(Auto)” which tells the users that the value of the cell is automatically generated.

- **Automatically generate key values for auto-increment fields.**

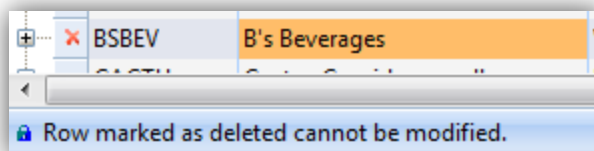
WebGrid includes ability to generate key values for auto-increment fields to simulate the database structure in the client side, which enables the pending changes concept to work effectively and reliably.

- **Select first cell after adding new row.**

WebGrid provides a new setting to focus the selection to first editable cell upon successful row adding operation, which significantly improves data entry process. This setting can be turned on by setting *SelectFirstCellOnAdd* to true in the *LayoutSettings* object.

	OrderID	ProductID
>	(Auto)	
	10410	33
	10410	59
+	(Auto)	10

- **Rows with *deleted pending change* can't be edited.**



When a row is marked with deleted pending change, WebGrid disables the row from further editing. This behavior is designated to prevent conflicts and unnecessary errors during the server-side batch update process.

- **New rows will always be added to the last position in the table.**

*Add pending changes* in WebGrid will always be shown consistently in the last position of the table according to the sequence of insertion. This design allows users to easily locate newly

+	ALFKI	11011	Janet
+	ALFKI	(Auto)	Nancy
+	ALFKI	(Auto)	Andrew

added rows whenever they need to access it. Furthermore, the added rows will be displayed regardless of the view settings such as in the condition of sorted, filtered or grouped.

- **Added row will be removed from view on delete.**

When delete action is performed on newly added row, WebGrid intelligently performs undo action on the specified new row. As a result, the added row will be removed from the client view as well as its associated pending changes.

In addition to the editing behaviors enhancement, SmartBatchUpdate™ also enhances the user experience in overall. To learn more, please see [automatic changes preservation](#) and [compatibility with existing features](#).

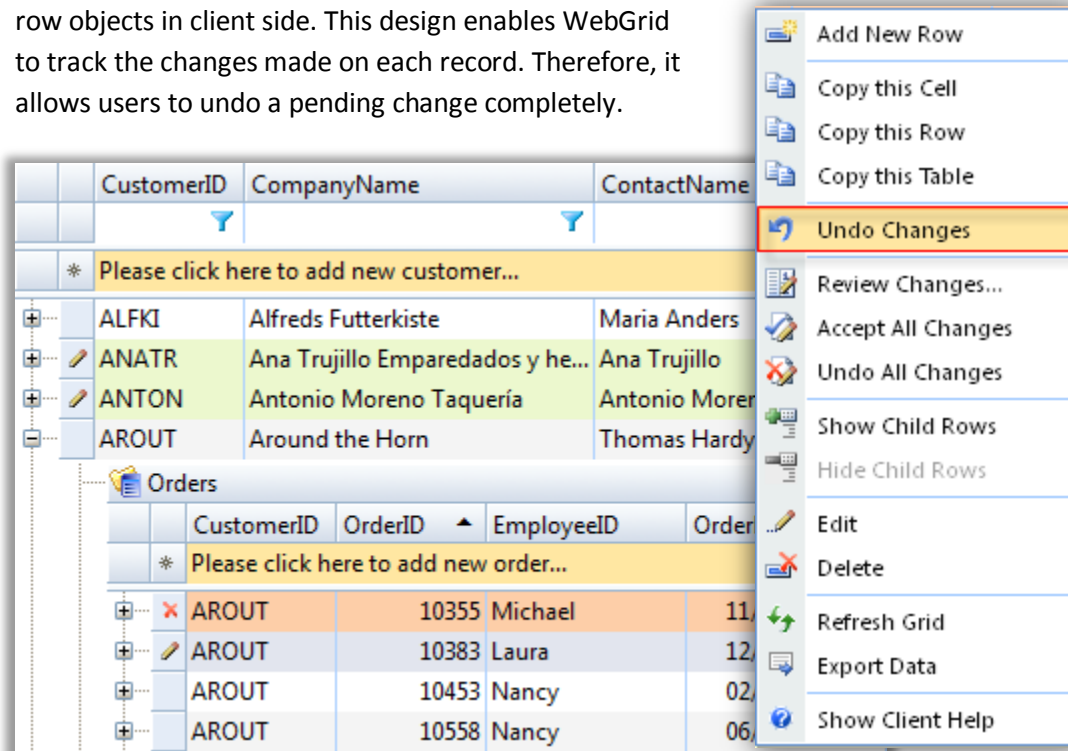
## Built-in Changes Management

SmartBatchUpdate™ provides a high-level built-in changes management, in addition to the solid underlying infrastructure which maintains the integrity and consistency of pending changes and batch update process.

Changes management in WebGrid Enterprise 7 includes the following features:

- **Undo Changes.**

WebGrid stores all pending changes and all its associated row objects in client side. This design enables WebGrid to track the changes made on each record. Therefore, it allows users to undo a pending change completely.



You can access *Undo Changes* command by bringing the row's context menu (right click on selected row). If you would like to undo all changes that you have made, you can click on Undo All Changes command in the row's context menu. Alternatively, you can easily locate the command in the status bar.

Please note that undo changes command will revert all changes back into its original state. Thanks to the state-of-the-art [pending changes architecture](#), the operation is done in real-time without the needs for server postback/callback.

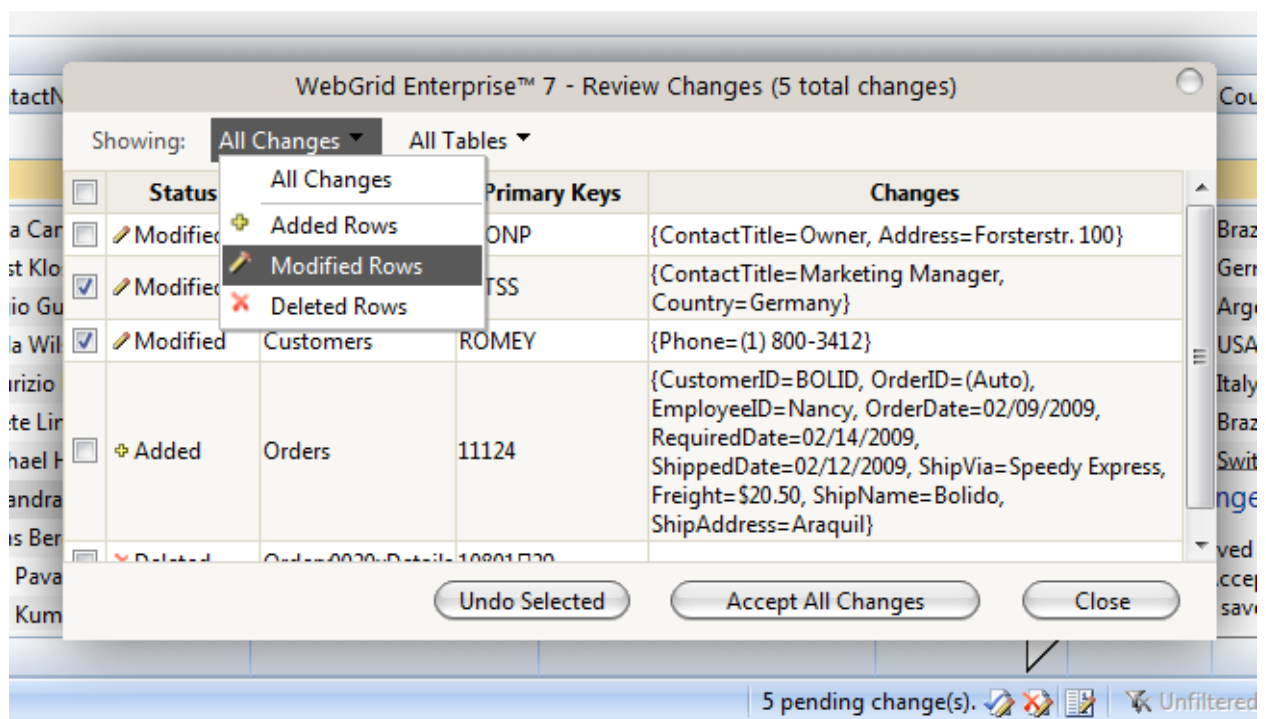
- **Review Changes.**

Designed with solid architecture, SmartBatchUpdate™ allows your end user to make dozens to hundreds of pending changes in a single session.

For instance, end user is allowed to make changes from one page, navigate to other page through paging function and make changes on the other pages, and so on. With so many changes in different views, users often have difficulty in reviewing or locating the pending changes.

To support these dynamic scenarios, users will need the ability to access all pending changes that they have made in different views. *Review Changes* is a powerful runtime feature that makes it easy and efficient for users to review all changes regardless of the tables and pending changes state.

*Review Changes* feature sports sleek dialog box interface to provide end user with a streamlined and convenient access to all pending changes within a single location. See below screenshot to get a better picture.



Review Changes dialog box provides easy-access to all pending changes across tables and views – makes it easy for users to undo several records or accept changes.

Note that the changes will be preserved even though the view has changed completely – eg, through sorting or paging. To learn more, see [Automatic Changes Preservation](#).

## Hierarchical Tables Support

SmartBatchUpdate™ includes full support for hierarchical tables configuration, makes it the most advanced and reliable solution for enterprise-class data editing requirements.

Consider a WebGrid with **Customers-Order-Order Details** configuration. Ideally, end user will need the ability to perform changes in child table in the same way and manner as they do it in root table.

WebGrid Enterprise 7's unique SmartBatchUpdate™ technology simply makes it happen. The following image shows a hierarchical WebGrid with several changes on each table. The pending changes can be easily recognized by its visual style and row header's indicator.

Customers						
CustomerID	CompanyName	ContactName	ContactTitle	Address	City	
* Please click here to add new customer...						
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	New Address	México D.F.	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London	
Orders						
CustomerID	OrderID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia
* Please click here to add new order...						
AROUT	10355	Michael	11/15/1996	12/13/1996	11/20/1996	Speedy Express
AROUT	10383	Laura	12/16/1996	01/13/1997	12/18/1996	Speedy Express
Order Details						
OrderID	ProductID	UnitPrice	Quantity	Discount	SubTotal	
* Please click here to add new order details...						
10383	Konbu	\$4.80	20	0%	\$96.00	
10383	Valkoinen suklaa	\$20.00	15	0%	\$195.00	
10383	Gnocchi di nonn...	\$30.40	20	0%	\$608.00	
(Auto)	Aniseed Syrup	\$20.00	10	0%		

Ready. 9 pending change(s). Unfiltered Page 1 of 4

When batch update feature is enabled in WebGrid, it will be automatically applied globally to include all child tables.

The batch update feature is supported in hierarchical WebGrid, regardless of the datasource type. That means no matter what your WebGrid is bound to – either it is a DataSet object, a hierarchical custom object, or an ISDataSource object – the batch update will always work in the same way and consistent fashion.

The batch update includes specific support for hierarchical WebGrid such as:

- **Maintain relations through referential integrity.**

WebGrid requires the tables to have valid referential integrity, that's all it needs for batch update feature to work. The referential integrity is usually defined at DataSet level through Relationships object. WebGrid makes use of this information in hierarchical data processing as well as in batch update processing.

Through referential integrity between each table, WebGrid will be able to store appropriate changes such as edits or deletes in the child table.

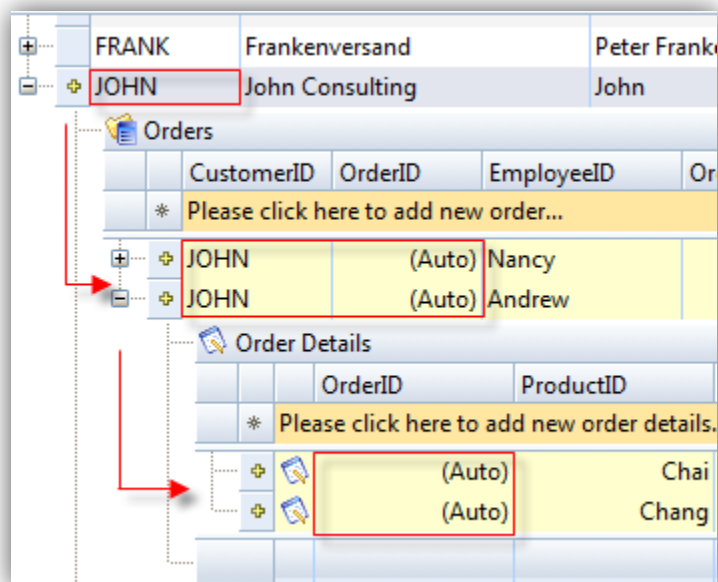
- **Cascading Inserts.**

One of the most challenging requirements in hierarchical editing is the ability to perform cascading inserts in batch update mode.

Cascading inserts essentially means that newly added rows in each linked table should be submitted at the same time in a single update process. From end user point, they require the ability to create the new row in parent table and continue to add its child rows – without having to physically create the parent row in the server.

SmartBatchUpdate™ is designed to fully support *cascading inserts*, making it easier and more efficient for end user to add new data across multiple child tables.

An example of cascading inserts in hierarchical WebGrid can be seen in the following illustration.



As illustrated in the above image, user can quickly add a new data in the *Customers* table, drill down the record (even though it's not existed in server yet), add two new records into *Orders* table, drill down the Order record again, and finally add records to the *Order Details* table.

The cascading inserts is made possible as SmartBatchUpdate™ smartly virtualizes the referential links and structure of each table, and intelligently generates the unique auto-generated values in the client side. See the red-boxed hints in above illustration.

WebGrid supports auto-generated values for both auto-increment data type such as Int32 and GUID type. WebGrid will automatically show *(Auto)* text for auto-increment fields.

When it comes to server-side processing, i.e. when user invoked *Accept Changes* command, WebGrid will automatically translate the new identity with the one resulted from the database inserts and map it recursively to the linked child rows. For more information, please see [Updating to Physical Database](#).

## Automatic Changes Preservation

As in the good tradition of WebGrid, SmartBatchUpdate™ is rigorously designed for the best user experiences. In simpler words, the batch update feature shouldn't limit end user to perform common data operations – especially the one that completely changes the current view.

For quick instances – data sorting, grouping or filtering are normally disabled when batch update feature is used. Some common reasons are that the changes are either too difficult to be maintained, or impossible to be preserved between view change.

SmartBatchUpdate™ takes data editing to a new height by allowing users to do what they used to do – column sorting, filtering, grouping, and even data paging – while simultaneously maintain the existing pending changes that they have made.

WebGrid does not only maintain the existing pending changes when you work on different views of your data, it perfectly restores the visual styles and row state of your pending changes as in where it left off.

*Automatic changes preservation*, an integral feature of SmartBatchUpdate™ – enables many scenarios not possible to achieve in the past – dramatically enhances user productivity. For instance, it is possible to edit a record in page 1, then jump to page 3 and edit more records, then go back to page 1 and edit more fields on existing edited record.

Automatic changes preservation works on data operations that change the view completely, such as:

- Column Sorting
- Column Filtering
- Column Grouping
- Column Visibility/Position Changing
- Refresh and RefreshAll
- Child Table Loading
- Classic Paging
- Virtual Load Paging

This feature is enabled by default when batch update feature is enabled. There are no additional steps required by developers to activate this feature.

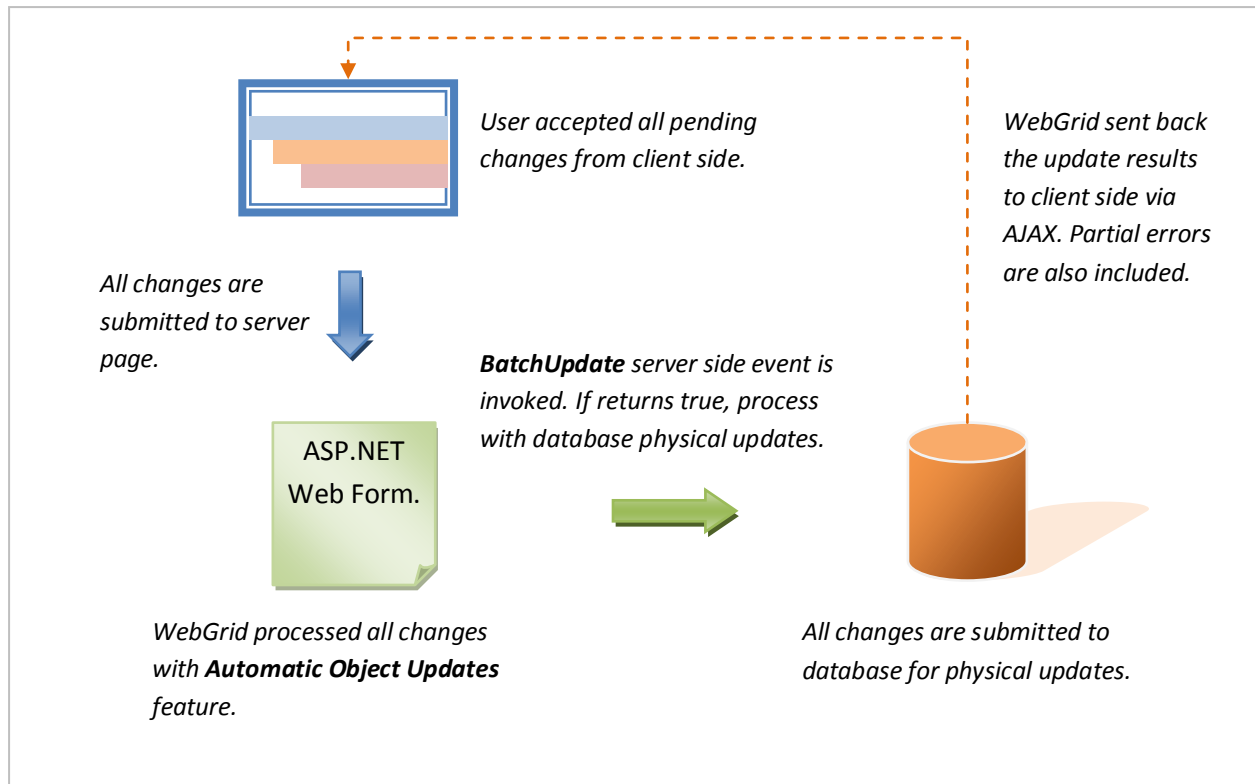
## Updating to Physical Database

SmartBatchUpdate™ provides sophisticated implementation to automate the batch updating process to the physical database.



The automatic updating feature significantly reduces development time – as you are not required to write any codes to perform the batch update, or very minimal efforts when you need to customize the updating process in more advanced scenarios.

To understand the physical updating concept better, please see the following illustration.



### Understanding Batch Update Processes

As shown in the illustration above, the physical update has several processes such as detailed in the following:

- **Automatic object updates.**

This setting is enabled by default. This feature will attempt to automatically apply the submitted pending changes to the intermediate data source that hold the objects during the binding process.

For instance, when WebGrid is bound to DataSet or DataTable, your pending changes will be applied and then mapped to your data source. This enables you to simply call a line of code to perform the physical updates via DbAdapter.

In more advanced scenario, such as when WebGrid is bound to unstructured data source or custom object, you can disable this feature by setting the *AutomaticObjectUpdates* property in *BatchUpdateSettings* to *false*.

In addition to single table support, this feature is also designed to support nested hierarchical tables that linked through **Referential Integrity**. This feature makes advanced scenarios possible such as [cascading inserts](#) and other scenarios related to hierarchical tables.

Please note that WebGrid doesn't perform *physical update* in this process.

- **BatchUpdate server side event.**

WebGrid provides a new server side event named *OnBatchUpdate*, which is invoked when the pending changes are required to be submitted to physical database.

When bound to data source other than data source controls, developers can handle *OnBatchUpdate* server side event to write the codes required to update the changes into physical database.

When bound to updatable data source controls – such as *AccessDataSource*, *ObjectDataSource*, and others – WebGrid will handle all physical updates automatically, given that *ReturnValue* is *true* in *OnBatchUpdate* event. The *ReturnValue* is *true* by default, which can be set to *false* to cancel automatic physical updates.

*OnBatchUpdate* server side event provides *BatchUpdateEventArgs* in the event argument, which is useful for developers who would like to customize the physical updating process, such as in the case of custom object binding.

The *BatchUpdateEventArgs* contains two properties:

- **PendingChanges.** Returns a *List<WebGridRowChanges>* object.  
Access this property to get all pending changes, regardless of the row state and tables. For more information about the object model, see [Server-side Programmability](#)
- **ReturnValue.** Returns a *Boolean* value.  
When bound to updatable data source controls, such as *SqlDataSource*, WebGrid automatically connects to the data source controls and invokes the update function to process the changes. You can set this property to *false* to prevent WebGrid to continue the automatic updates, such as in the case of failed validation or other business logic constraints.

The following C# codes show how to update all changes in a flat WebGrid that bound to a *DataSet* object.

```
void WebGrid1_BatchUpdate(object sender, BatchUpdateEventArgs e)
{
    CustomersTableAdapter daCustomer = new CustomersTableAdapter();
    dsNorthwind ds = (dsNorthwind) WebGrid1.GetCachedDataSource();
```

```

    daCustomer.Update(ds); // updates all changes to database
}

```

- **Partial errors support.**

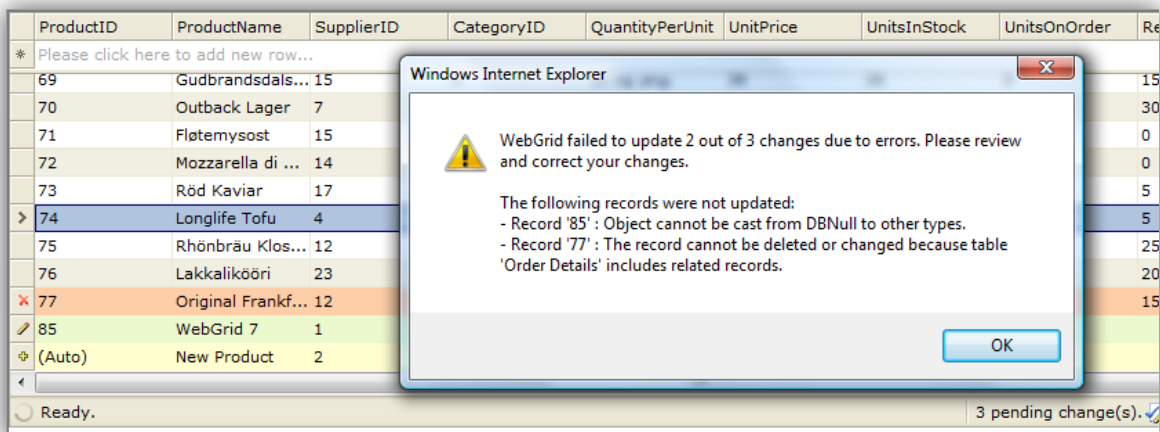
In addition to solid [batch update architecture](#) and automatic object updates, SmartBatchUpdate™ is also equipped with support for partial errors, making it the most advanced and reliable solution for client-side data editing application.

Partial errors occurred when one or more changes are failed to be updated while there are also some successful updates. Partial errors will not occur when all changes are failed.

With partial errors support, user can make changes with confidence, without have to worry that error in one of the changes will cause all changes to fail. This means that WebGrid is able to isolate erroneous changes, and continue to update the next changes that are unrelated to the previously failed changes update.

At client-side, WebGrid also restores the pending changes that were not successfully updated, so that end user can conveniently correct the errors and accept the changes.

The following image shows a WebGrid with partial errors response. The error detail for each failed updates are shown in the message box, making it easy for end user to review and revise the errors.



For more control over partial errors response in the client side, you can handle the **OnBatchUpdateSuccess** client side event and access the *rowErrorXml* parameter to get the error detail on each failed update. For more information about client side usage, please see [Client-side Programmability](#)

## Various Datasource Support

WebGrid Enterprise 7 supports physical database updates in various scenarios, such as when bound to different type of datasource, hierarchical tables configuration, and more. The following list describes the supported scenarios in more details:

- **Traditional Binding (ADO.NET DataSet and DataTable)**

When bound to ADO.NET-compatible data source such as DataSet and DataTable, you will need to write codes in *OnBatchUpdate* server side event to update the changes to database.

However, the required codes should be very minimal as ADO.NET already implemented batch update at data access level. Developers will then be able to simply invoking a single Update method to process all changes, which were previously mapped during [Automatic Object Updates](#) process.

The following C# codes show how to update all changes that bound to a DataTable object.

```
void WebGrid1_BatchUpdate(object sender, BatchUpdateEventArgs e)
{
    CustomersTableAdapter daCustomer = new CustomersTableAdapter();
    DataTable dt = (DataTable)WebGrid1.GetCachedDataSource();

    daCustomer.Update(dt); // updates all changes to database
}
```

- **Declarative Binding (Datasource Control such as SqlDataSource)**

Data source control is the most ideal data binding concept in ASP.NET that provides clear abstraction between UI and data logic. Introduced in .NET 2.0, data source control allows you to bind data in declarative markup, saving you from tedious tasks and lengthy codes.

SmartBatchUpdate™ takes advantage of data source control to the fullest. When you connect WebGrid to an updatable data source control, you don't need to write codes to handle the physical updates.

To enable batch update in declarative binding scenario, simply set *AllowBatchUpdate* to true and make sure your data source control has been configured properly to support data updates.

- **Hierarchical Traditional Binding (ADO.NET DataSet)**

Similar to Traditional Binding, you are also required to handle *OnBatchUpdate* server side event to write codes to perform database updates.

Thanks to the [automatic identity's mapping](#) through referential integrity, WebGrid performs all the complex logics behind the scene, so that you only need to write a few lines of codes to update the dataset.

The following C# codes show how to update all changes in each table contained in the DataSet object.

```
void WebGrid1_BatchUpdate(object sender, BatchUpdateEventArgs e)
{
    CustomersTableAdapter daCustomer = new CustomersTableAdapter();
    OrdersTableAdapter daOrders = new OrdersTableAdapter();
    Order_DetailsTableAdapter daOrderDetails = new
        Order_DetailsTableAdapter();
    dsNorthwind ds = (dsNorthwind)WebGrid1.GetCachedDataSource();

    // updates all changes per table adapter from the root table
    // to each child table in ordered sequence
    daCustomer.Update(ds);
    daOrders.Update(ds);
    daOrderDetails.Update(ds);
}
```

- **Hierarchical Declarative Binding (ISDataSource)**

As in flat WebGrid declarative binding, SmartBatchUpdate™ supports hierarchical WebGrid that is bound to [ISDataSource](#) control in the same way and consistent fashion.

With ISDataSource control, you are not required to write codes in order to perform physical update into the underlying database.

It is important to ensure that your ISDataSource instance has been properly configured to return new identity for each table in the event of insert. For more information, please see [New Identity Insert](#) in the section below.

- **Custom Object**

In addition to built-in .NET data sources, SmartBatchUpdate™ is rigorously designed to support advanced enterprise scenarios, such as using the feature in conjunction with custom object data binding.

When bound to custom object collection – such as a list of Customer objects – you can disable [automatic object updates](#) feature. In this case, you are responsible to write codes to perform physical updates according to your business logic/model.

In the same way as in other data sources, you handled *OnBatchUpdate* server side event to provide codes to perform physical update.

To learn more on how to use batch update feature in custom object scenario, please see [Server-side Programmability](#)

## New Identity Insert

One of the best practices in database design concept is to have at least one unique, auto-increment field in each table which acts as the row's identifier.

When you insert a new row to the table, the database will issue a new, unique identity value of the new inserted row. However, each database provider has different ways to retrieve the new identity's value. As such, Microsoft .NET Framework doesn't include built-in implementation to retrieve identity insert in its ADO.NET technology.

SmartBatchUpdate™ relies heavily on row's identifier to differentiate each record in order to work properly. Therefore, developers should make sure that the new identity's value is correctly retrieved and passed to WebGrid during batch update processing.

The following list describes the techniques to retrieve the new identity's value in various data source type:

- **ADO.NET (DataSet).**

ADO.NET, as the middle-tier data object provided in Microsoft .NET, does not contain database specific implementation in its data access level. As such, the design enables it to work with any database providers that support .NET Framework.

When a new row is inserted through DbAdapter compatible provider, it does not return the new row's identity by default. Thus, developers are required to write additional codes to programmatically obtain the new row's identity.

The following C# codes show how to retrieve the new identity for Access or Sql database provider.

```
void WebGrid1_BatchUpdate(object sender, BatchUpdateEventArgs e)
{
    CustomersTableAdapter daCustomer = new CustomersTableAdapter();
    OrdersTableAdapter daOrders = new OrdersTableAdapter();
    Order_DetailsTableAdapter daOrderDetails = new
        Order_DetailsTableAdapter();
    dsNorthwind ds = (dsNorthwind)WebGrid1.GetCachedDataSource();

    // Handle RowUpdated event of DataAdapter
    daOrders.DataAdapter.RowUpdated += new
        OleDbRowUpdatedEventHandler(DataAdapter_RowUpdated);

    daCustomer.Update(ds);
    daOrders.Update(ds);
    daOrderDetails.Update(ds);
}
```

```

void DataAdapter_RowUpdated(object sender, OleDbRowUpdatedEventArgs e)
{
    // Conditionally execute this code block on inserts only.
    if (e.StatementType == StatementType.Insert)
    {
        if (e.Row.Table.TableName == "Orders")
        {
            OleDbCommand cmdNewID = new OleDbCommand("SELECT @@IDENTITY",
                e.Command.Connection);

            /* Retrieve the new identity and call UpdateRowIdentity to
               process the new identity. */

            WebGrid1.GetTableByName("Orders").UpdateRowIdentity("OrderID",
                e.Row["OrderID"], cmdNewID.ExecuteScalar());
        }
    }
}

```

#### Codes Explanation:

1. In *OnBatchUpdate* server side event, you write your codes that invokes physical database updates. In the sample above, the *RowUpdated* event is handled before any calls to *Update* method.
2. The function delegate for *RowUpdated* will be invoked for each successful insert. The above codes perform checking on *StatementType* and *TableName* respectively.
3. A new *OleDbCommand* object is created to perform select identity retrieval, based on existing connection instance. It's important that you use the existing connection instance in order to get the new identity properly. Again, note that the select identity retrieval statement above only works for Access and Sql-compatible provider.
4. Finally, *UpdateRowIdentity* method of the respective *WebGridTable* instance is called, passing the identity's data member name, the current value of the *OrderID*, and the new identity's value resulted from the successive insert.

It's extremely important to call *UpdateRowIdentity* method to pass in the obtained new identity value. The method maps the new identity into each inserted row properly, which enables WebGrid to process the changes efficiently specifically in hierarchical configuration and more advanced scenarios such as [cascading inserts](#).

- **Data source control.**

Most data source controls that shipped with Visual Studio® 2005 or 2008 do not support automatic identity retrieval. The only data source control that supports automatic identity retrieval is *SqlDataSource*.

WebGrid extends the data source control further by providing automatic identity retrieval support for *AccessDataSource*, in addition to *SqlDataSource*. This means that if you bound WebGrid to either *AccessDataSource* or *SqlDataSource*, you don't need to write additional codes to retrieve the new identity.

The following C# codes show how to retrieve new identity for *ObjectDataSource* control. The sample is referencing to Orders table in Northwind sample database.

```
[DataObjectMethodAttribute(DataObjectMethodType.Insert, true)]
public int DoInsert(ref Nullable<int> OrderID, string CustomerID,
    Nullable<int> EmployeeID, Nullable<System.DateTime> OrderDate,
    Nullable<System.DateTime> RequiredDate, Nullable<System.DateTime>
    ShippedDate, Nullable<int> ShipVia, Nullable<decimal> Freight, string
    ShipName, string ShipAddress, string ShipCity, string ShipRegion,
    string ShipPostalCode, string ShipCountry)
{
    this.Connection.Open(); // important
    int affectedRows = this.Insert(CustomerID, EmployeeID, OrderDate,
        RequiredDate, ShippedDate, ShipVia, Freight,
        ShipName, ShipAddress, ShipCity, ShipRegion,
        ShipPostalCode, ShipCountry);

    int? identity = this.SelectIdentityQuery();

    OrderID = identity; // pass the new identity into OrderID param
    this.Connection.Close();

    return affectedRows;
}

public int SelectIdentityQuery()
{
    OleDbCommand cmd = new OleDbCommand("SELECT @@IDENTITY",
        this.Connection);
    int newID = (int)cmd.ExecuteScalar();

    return newID;
}
```

At the WebForm (ASPX) page, make sure you reflect the *InsertCommand* with the new extended method name above. Furthermore, you'll need to add a new *OrderID* parameter in the *InsertParameters* collection. See the following sample:

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server" ...
    InsertMethod="DoInsert" OnInserted="ObjectDataSource1_Inserted">
    <InsertParameters>
        <asp:Parameter Direction="InputOutput" Name="OrderID"
            Type="Object" />
        ...
    </InsertParameters>
</asp:ObjectDataSource>
```



At the page code behind, you will need to provide the function as the delegate for *OnInserted* event that is defined in the WebForm (ASPX).

The following C# codes show how to access the output parameters that obtained during *DoInsert* command, and pass it to WebGrid for further processing.

```
protected void ObjectDataSource1_Inserted(object sender,
                                         ObjectDataSourceStatusEventArgs e)
{
    WebGrid1.SetOutputParameters(e.OutputParameters);
}
```

Codes Explanation:

1. By default, the Visual Studio-generated table adapter doesn't contain implementation to return new identity upon successive inserts. Thus, we created a new method to extend the table adapter's Insert functionality.
2. A new parameter *OrderID* with ref keyword is inserted into the first parameter of the extended Insert method. This parameter is used to hold the new identity value which is assigned during the process.
3. The *SelectIdentityQuery* method is used to obtain the row's new identity as the result of successive insert. This query is compatible with Access and Sql providers.
4. The extended Insert method name is assigned to the *InsertCommand* of the *ObjectDataSource*.
5. Handle *OnInserted* server side event of the *ObjectDataSource* to obtain the resulted output parameters and pass it to WebGrid for further processing.
6. Finally, adds a new *<asp:Parameter>* object into the *InsertParameter* collection of the *ObjectDataSource*. This new parameter is required to reflect the parameter signature of *DoInsert* method, which is used to hold the new identity value of *OrderID* column.

- **Intersoft Datasource Control (ISDataSource)**

Intersoft's flagship *ISDataSource* control also works the same way and manner as *ObjectDataSource*, where it is serving as intermediate data class that bridge between User Interface and backend provider. As such, *ISDataSource* does not contain vendor-specific implementation in order to support various database providers supported by .NET Framework.

*ISDataSource* is the only provider in the market that supports multiple tables. As the result, WebGrid accepts only *ISDataSource* control for its *hierarchical table* feature.

To retrieve new identity in *ISDataSource*, you can use the same technique as in *ObjectDataSource* above. The only exception is that you can skip the fifth step above. *ISDataSource* doesn't require you to pass the output parameters to WebGrid, since it is tightly integrated with *ISDataSource* and thus it has exclusive capability to access *ISDataSource* instance through standardized interfaces.

## Localization

The new batch update feature in WebGrid Enterprise 7 introduces a dozens of new [User Interface elements](#), which contains many new textual settings as well.

As in good tradition of WebGrid, all textual settings are customizable according to the culture specified by developer. Each textual setting can also be overridden individually. You can find the complete text settings in default.xml which is located in the Localization folder.

The following table lists the new text settings that you can customize.

Category	Text Setting Key	Default Text Setting Value (English)
CommonText	PendingChanges	{0} pending change(s).
	DeletedRowNoEdit	Row marked as deleted cannot be modified.
	DeletedRowNoExpand	Row marked as deleted cannot be expanded.
	BatchUpdateSuccess	All changes have been successfully updated to server.
	BatchUpdateSuccessWithPartialErrors	One or more changes are not updated due to server errors.
MessageBoxText	UnsavedPendingChanges	You have made one or more changes on WebGrid records. Any unsaved changes will lose.
	NotifyPendingChanges	There are {0} unsaved pending changes. Click Accept Changes button below to save changes.
	UndoPendingChanges	Are you sure you want to undo all pending changes?
	UndoSelectedPendingChanges	Are you sure you want to undo selected pending changes?
	DeletedRowEditException	You cannot make changes to the row which parent has been marked as deleted.
	DuplicateKeyException	The pending changes already contain a record with key '{0}'
	PrimaryKeyEditException	Modification on primary key's field of pending changes is not allowed.
	BatchUpdateException	An exception has occurred while processing batch update. Please correct or undo your changes.\n\nException details:\n{0}
	BatchUpdatePartialException	WebGrid failed to update {0} out of {1} changes due to errors. Please review and correct your changes.\n\nThe following records were not updated:\n{2}

Tooltip	ModifiedRowState	Modified row
	AddedRowState	Added row
	DeletedRowState	Deleted row
	PendingChangesEmpty	There are no pending changes
	PendingChangesExisted	There are {0} pending changes: {1} insert(s), {2} edit(s), {3} delete(s)
	ReviewChanges	Review Pending Changes
	UndoChanges	Undo All Changes
	AcceptChanges	Accept All Changes
ContextMenu/Row	UndoSelection	Undo Selection
	DeleteSelection	Delete Selection

Note: You may find empty text entries when using languages other than English. If you found empty entries on the language and culture that you are familiar with, please help us to translate the entries using the default.xml as the template and send it to [feedback@intersoftpt.com](mailto:feedback@intersoftpt.com).

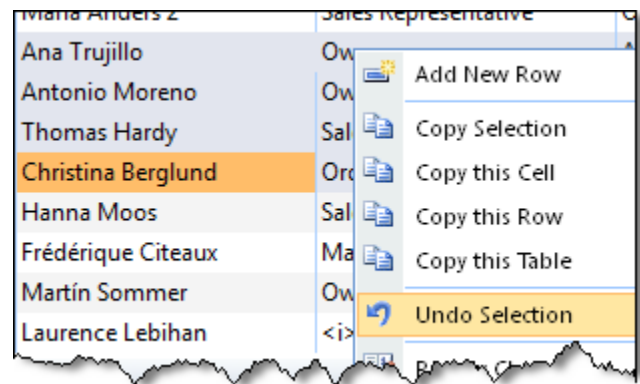
## Compatibility with Other Features

As in every major release of WebGrid Enterprise, every new feature and enhancement are designed to work in conjunction with existing features. There are, though, some conditions and scenarios where the enhancements are not applicable, such as due to behavior conflicts.

In addition to comprehensive [hierarchical tables](#) and [client side editing](#) support, SmartBatchUpdate™ is also designed to work with existing features. Some of noteworthy key features are such as listed in the following:

- All user interaction features, such as column resizing, moving, selection and context menu.
- All databound operations, such as sorting, grouping and filtering.
- Special support for column grouping, where newly added rows will remain visible and will be properly indented according to the group level.
- Data paging includes both classic paging and virtual load.
- Cell-select editing mode.
- Columnset layout mode.
- Column freezing.
- Auto-filter suggestion.
- Preview row.
- Self referencing.

*Multiple Selection* feature is also enhanced to work in harmony with batch update feature. When you select multiple rows that contain pending changes, the **Undo Selection** command will appear in context menu, allowing end user to quickly undo multiple pending changes. Also, when deletion feature is enabled and multiple rows are selected, **Delete Selection** will appear in the context menu.



All client side APIs and object models continue to work and function normally as published in the documentation.

There are no unknown compatibilities with specific features that could cause errors or misbehaviors when batch update is enabled as far as the time of this writing.

## Client-side Programmability

SmartBatchUpdates™ is a set of comprehensive object models and application programming interfaces (API), serving as the main foundation and architecture for the [pending changes concept](#) introduced in WebGrid Enterprise 7.

All pending changes operation that invoked from WebGrid's user interface is also using the same set of APIs and interfaces. The result is a powerful and highly extensible client side batch editing architecture, which enables developers to further extend the functionality to address their complex business requirements.

The following sections describe the new classes, methods and client side events that introduced as core programming interfaces for the batch update feature.

### New classes

- **WebGridBatchUpdateSettings.** This class contains all settings and behaviors related to batch update feature.

Properties:

- AllowReviewChanges (bool)
- AllowUndoChanges (bool)
- AutomaticObjectUpdate (readonly, bool)
- PromptOnUndoAllChanges (bool)
- NotifyOnLostFocus (bool)
- HighlightChanges (bool)

- **WebGridCellData.** This class holds the delta information of modified cell data.

Properties:

- Column (WebGridColumn object)
- OldValue (object)
- OldText (string)
- NewValue (object)
- NewText (string)

- **WebGridRowChanges.** This class represents the changes for a row object.

Properties:

- RowState (readonly, string). Possible values: Added, Modified, Deleted.
- Element (readonly, HTMLRow object)
- Data (readonly, Array of WebGridCellData)
- Row (readonly, WebGridRow object)

- KeyValues (readonly, string)
- Table (readonly, WebGridTable object)

Methods:

- MergeChanges. Parameter: the row to be merged (WebGridRow object).
- GetExistingData. Parameter: the name of cell to retrieve (string).

### New methods

Several methods have been added as extensions to provide batch update functionality. These methods will only be available when batch update feature is used.

Class	New Method	Parameters
<b>WebGrid</b>	GetChangesCount (Returns the count of all pending changes regardless of the state and table)	-
	UpdatePendingChangesStatus (Synchronize pending changes with Grid's user interface elements)	-
	UndoAllChanges (Undo all pending changes regardless of the state and table)	-
	AcceptAllChanges (Accept all pending changes and submit them to server for batch update)	-
	GetChanges (Returns a list of WebGridRowChange objects)	rowState (Specifies the state of the changes to get)
	InvalidateChanges (Invalidate all pending changes to be reprocessed in the next synchronization)	-
	ClearChanges (Clear and remove all pending changes regardless of the state and table)	-
<b>WebGridTable</b>	GetChanges (Returns a list of pending changes that existed in this table)	rowState (Specifies the state of the changes to get)
	GetChangesCount (Returns the count of the pending changes in this table)	-
	UndoChanges (Undo all pending changes in this table)	-
	InvalidateChanges (Invalidate all pending changes in this table)	-
<b>WebGridRow</b>	GetChanges (Returns the changes made to this row)	rowState (Specifies the state of the changes to get)
	GetRowState (Returns the state of this row. It will return Unmodified if this row has no changes)	-
	InvalidateRowState (Invalidate the state of this row to be processed in the next synchronization)	-
	UndoChanges (Undo changes made to this row)	skipUpdateStatus (Specifies whether to skip the status updating after undo)

	AddPendingChanges (Mark changes made to this row and add it into pending changes collection)	-
	UpdatePendingChangesUI (Synchronize the pending changes data with Grid's user interface elements)	isProgrammatic (Specifies whether the method is called by user code)
	SetDeleted (Mark this row as deleted)	-
<b>WebGridCell</b>	SetChanges (Change the text and value of this cell)	text (The new text to be set) value (The new value to be set. This parameter is optional, and only useful for valuelist column)

### New client side events

The following table describes the new client side events related to batch update feature.

Event Name	Description	Parameters
<b>OnUndoChanges</b>	Fired when a row changes is about to be undo'ed.	controlId, rowObject (The row to be undo'ed)
<b>OnUndoAllChanges</b>	Fired when undo all changes command is performed.	controlId
<b>OnAcceptAllChanges</b>	Fired when accept all changes command is performed.	controlId
<b>OnAddPendingChanges</b>	Fired when a pending changes is about to be added.	controlId, table (The table on which pending changes is added to), rowChange (The row change object)
<b>OnRemovePendingChanges</b>	Fired when a pending changes is about to be removed.	controlId, table (The table on which pending changes is added to), rowChange (The row change object)
<b>OnBatchUpdateSuccess</b>	Fired when batch update operation has been successfully completed.	controlId, hasPartialErrors (Whether the batch update contains one or more errors), partialErrorsXml (The xml document contains partial errors data)

## Server-side Programmability

As in client-side, the same set of new classes and methods have also been added to server-side. This enables reliable synchronization between client-side and server-side, which results in consistent interfaces and elegant objects design.

For shared set of the new classes, methods and properties; please refer to [Client-side Programmability](#) section above.

The following list shows additional methods to perform server-side related tasks.

- **WebGrid class.**

Methods:

- *GetChanges()*  
Returns all pending changes regardless of the row's state and table.
- *GetChanges(RowState rowState)*  
Returns all pending changes with given row state.
- *PerformBatchUpdate(bool throwExceptionOnError)*  
Processes changes and updates it to physical database programmatically.
- *GetBatchUpdateExceptions()*  
Returns one or more exceptions that occurred while processing batch update.
- *AddBatchUpdateException(WebGridBatchUpdateException exception)*  
Adds an exception when a batch update-related error occurred. Use this method to take advantage of partial error support in custom object updating scenario.

Events:

- *OnBatchUpdate*. Provided *BatchUpdateEventArgs* as the event argument.  
This event will always be invoked when batch update feature is enabled. This event is required to be handled for traditional and custom object binding. You can cancel WebGrid for further processing by setting false to its *ReturnValue*.

- **WebGridTable class.**

Methods:

- *GetChanges(RowState rowState)*  
Returns all pending changes in this table, given the row state.
- *UpdateRowIdentity(string dataMember, object originalValue, object processedValue)*  
Updates and maps new identity of successive row inserts to intermediate objects in the data source. This method is required for traditional and custom object binding.



The provided server side APIs, as described above, enables you to programmatically work against pending changes data that submitted from client-side.

Several scenarios that can be achieved by using the provided APIs:

- **Get the pending changes per table and per row state for further processing in your business object level.**

You can use *GetChanges* method at *WebGrid* or *WebGridTable* object to get the desired changes.

The *RowState* enumeration includes the following values: Added, Modified and Deleted. You can query the changes based on one or multiple *RowState*.

For example, the following C# codes show how to query pending changes for Added and Modified state:

```
List<WebGridRowChanges> changes =
    WebGrid1.RootTable.GetChanges(RowState.Added | RowState.Modified);

foreach (WebGridRowChanges change in changes)
{
    // process each changes
}
```

Note that the pending changes data will not be submitted on WebGrid's FlyPostBack™ actions for performance wise. The changes will be available upon the following conditions:

- Full Postback.
  - *Accept Changes* FlyPostBack™ action.
  - ASP.NET AJAX callback.
- **Perform batch update on a server-side button click. In some cases, you might prefer to have your own button to process the pending changes, in addition to the user interface provided by WebGrid.**

You can achieve this task by calling *PerformBatchUpdate* method available in *WebGrid* object. See the following C# sample:

```
private void Button1_Click(object sender, EventArgs e)
{
    WebGrid1.PerformBatchUpdate(true);
}
```

The *PerformBatchUpdate* follows the same process and life cycle of batch update, such as processing [automatic object updates](#) and invoking *OnBatchUpdate* event.

- **Process pending changes programmatically and call your custom data access method to perform physical update. This scenario is especially useful when you bind WebGrid to custom object collection instead of dataset or datasource control.**

The following C# codes show how to use many of the server-side methods to iterate each change and update it using custom data access layer.

```
protected void WebGrid1_BatchUpdate(object sender, BatchUpdateEventArgs e)
{
    foreach (WebGridRowChanges rowChanges in e.PendingChanges)
    {
        CustomObjects.Customer customer = null;

        try
        {
            switch (rowChanges.RowState)
            {
                case RowState.Added:
                    customer = new CustomObjects.Customer();
                    MapChangesToObject(rowChanges.Data, customer);
                    DataLayer.InsertCustomer(customer);
                    break;

                case RowState.Modified:
                    customer =
                        DataLayer.GetCustomer(rowChanges.KeyValue.ToString());

                    MapChangesToObject(rowChanges.Data, customer);
                    DataLayer.UpdateCustomer(customer);
                    break;

                case RowState.Deleted:
                    customer =
                        DataLayer.GetCustomer(rowChanges.KeyValue.ToString());
                    DataLayer.DeleteCustomer(customer);
                    break;
            }
        }
        catch (Exception exp)
        {
            // adds the Exception to batch update list to take advantage of
            // WebGrid's partial errors support.

            WebGrid1.AddBatchUpdateException(
                new WebGridBatchUpdateException(rowChanges,
                    new Exception("Record '" + rowChanges.KeyValue.ToString() +
                        "' has error '" + exp.Message + "'", exp)));
        }
    }
}
```

The following is the C# codes to map the changes into object.

```
private void MapChangesToObject(List<WebGridCellData> data,
    CustomObjects.Customer customer)
{
    foreach (WebGridCellData cellData in data)
    {
        string newText = cellData.NewText;
```

```

switch (cellData.Column.DataMember)
{
    case "CustomerID":
        customer.CustomerID = newText;
        break;

    case "Address":
        customer.Address = newText;
        break;
    ...
}
}
}

```

#### Codes Explanation:

- In custom object scenario, developers often have their own data access layer to perform data management such as select, inserts, updates and deletes. Thus, you need to handle *OnBatchUpdate* server side event to process the changes as shown in above sample codes.
- You can access all pending changes from *e.PendingChanges*, which is *List<WebGridRowChanges>*.
- You iterate on each *WebGridRowChanges*. Then perform update based on the *RowState* value.
- Notice that we used strongly-typed custom object to perform data update in elegant fashion. For add and modified changes, you need to map the changes into the object accordingly. In the sample above, it is done with *MapChangesToObject* function.
- Call *AddBatchUpdateException* when an exception occurred. This enables the codes to continue updating the other records. This error handling is also taking advantage of [partial errors support](#).

## Client-side data binding

### Overview

WebGrid Enterprise™ 7 is strongly focused on enterprise web development which has been evolved to adopt new web technologies such as cloud computing, software-as-a-service (SaaS) and other Web 2.0 related technologies.

Featuring a rock-solid client binding architecture, WebGrid Enterprise™ 7 is the most revolutionary release yet which has undergone major revamp and re-engineered to meet the high performance standards demanded by today's web application.

The following sections discuss Intersoft's approach to client data binding, its new technologies and benefits, and how it resolves performance bottleneck without trading-off fundamental features.

### What is client-side binding?

Client-side binding is a mechanism that processes data operation and binding life-cycle entirely in the client-side.

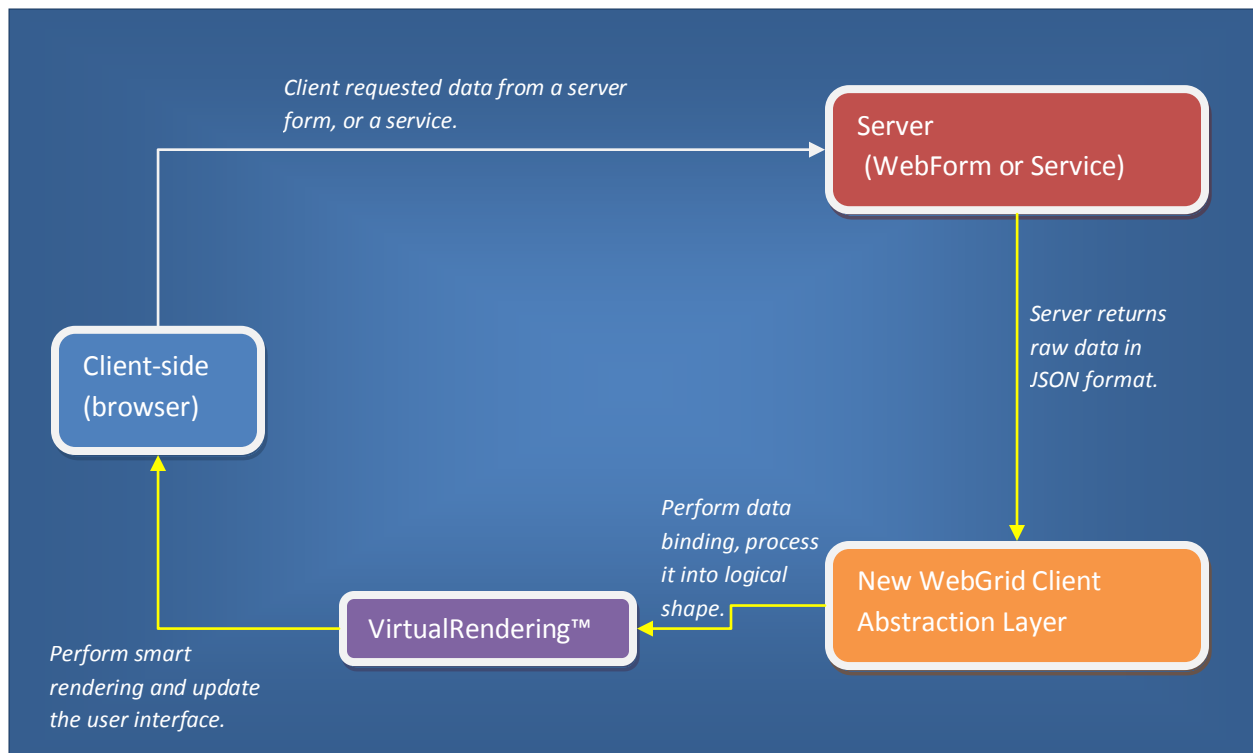
Data binding consists of a series of data processing operation that takes a raw-form of data and shapes it into a logical object, which is then rendered into user interface. Client-side binding simply means that data binding should be performed in the client-side (also known as browser), rather than in the server-side (ASP.NET).

All previous versions of WebGrid and most ASP.NET server-side controls perform data binding in the server-side since .NET Framework already offer rich base class libraries to perform these operations efficiently – such as data sorting, filtering, paging and other data access-related functions available in dataset, data table and data view.

Server-side binding normally sent HTML markup as the final output, which is directly recognized by browsers with very minimum overhead in the client side. In certain scenarios such as in data listing that consist of many rows and cells, the HTML markup result could become huge and takes longer time to transfer from the server to client. As a result, it could bring a potential performance problem that impact user experience in overall.

Unlike server-side binding, client-side binding often returns raw data which has significantly smaller size compared to HTML markup. WebGrid 7 incorporates new JSON (Javascript Object Notation) as its data exchange format for all data transfer operation required in client binding.

The following diagram illustrates the new WebGrid's architecture and client binding life cycle.



As seen in the above illustration, WebGrid Enterprise™ 7 is built on the top of solid client binding architecture which processes data operation entirely in the client-side – from the data dispatching, data shaping, binding to rendering.

#### Innovative VirtualRendering™

VirtualRendering™ is Intersoft’s state-of-the-art client rendering framework that enables WebGrid to render data rows efficiently, while at the same time delivering identical user interface as in server-side rendering.

This rendering technology is one of the key components in Intersoft’s client binding architecture that transform logical data object into user interface elements.

The uniqueness of Intersoft’s rendering technology lies in its ability to perform rendering with very minimum overhead. This is made possible with sophisticated state management that automatically synchronizes its current state as user interacts with the Grid.

As a result, WebGrid doesn’t require a row to be rendered from scratch which could lead to performance issues. Instead, it renders a row based on “delta algorithm” to produce high rendering quality in superior performance.

One of the outstanding achievements of VirtualRendering™ is its small size of the client scripts required to activate the client binding feature. Unlike traditional approaches that could bloat the client files up to half a megabyte, WebGrid 7 adds only as little as 60KB for data binding processing, data shaping and rendering.

## Benefits

Client-side binding provides comprehensive solution for developers to accomplish many advanced scenarios and tasks which were previously difficult or not possible to be achieved. The following list explains several key benefits of client-side binding:

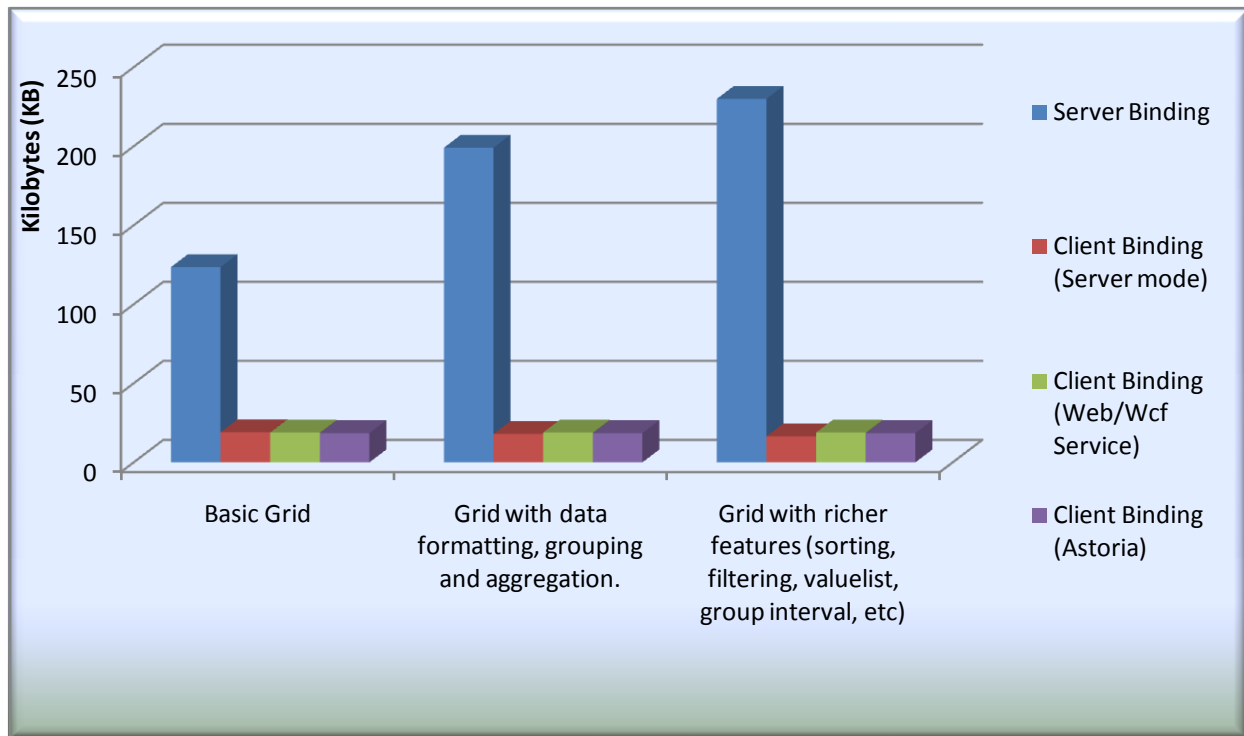
- **Data footprint reduction by over 90%.**  
When operating in client binding mode, WebGrid returns raw data in JSON format which is significantly more compact compared to HTML markup. In many scenarios, WebGrid has been proven to reduce data footprint consistently by over 90%. For more information, please refer to [Performance Benchmark and Comparison](#).
- **Superior performance and increased response time.**  
With re-engineered architecture to support client binding extensively, WebGrid yields better performance and increased response time as it skips several default behaviors, data processing and rendering that used to be produced from server side. Learn the [client binding concept](#) for more information.
- **More responsive, intuitive user experience.**  
The nature of client-side binding – which exchange data in its raw format, enables WebGrid to receive data faster from the server. As such, end user will get more responsive user experience even in relatively slower network connection.
- **Connect to external site or service-based datasource (“Cloud” support).**  
With solid client binding implementation, it enables developers to get the data from external location such as mashup, or from service-based datasource such as Web service, Windows Communication Foundation (WCF) service, as well as Microsoft’s new ADO.NET Data Service.
- **Future proof – supporting more service-oriented development methodology.**  
Intersoft’s approach to client-side binding is an agnostic implementation that completely independence from server-side object model. This enables WebGrid to perform data binding based on any raw form of data in JSON format, which can be produced by a server hosted by ASP.NET, PHP, or other platforms.

Based on the understanding of Intersoft’s client binding implementation, it also enables developers to achieve a completely unbound scenario as well as future development methodologies that are server-independence such as in ASP.NET MVC and Windows Azure™.
- **Build Web 3.0, offline-capable application.**  
WebGrid Enterprise™ 7 includes new features that designed to help you build advanced Web 3.0 application easier and faster. You can combine WebGrid’s state-of-the-art features such as client binding mode, in-line editing and [batch update](#) to build offline-capable application, which enables end user to make changes anywhere and anytime even without connection to server.

## Performance Benchmark and Comparison

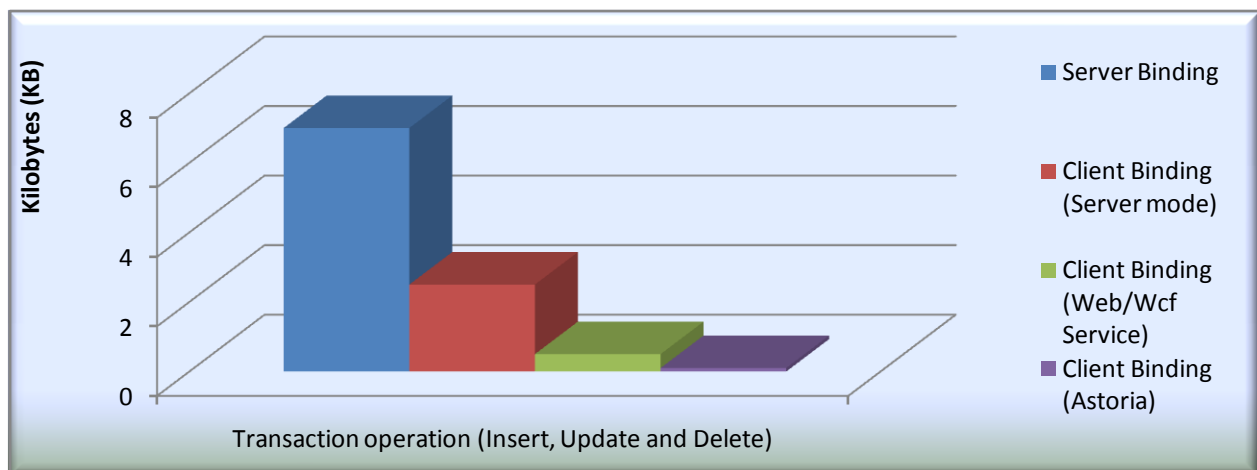
The following charts illustrate the performance comparison between server-side and client-side binding in several aspects, such as data footprint and network connection type. This section also shows the comparison between different modes of client-side binding.

### Data footprint size comparison on AJAX operation based on various Grid configurations



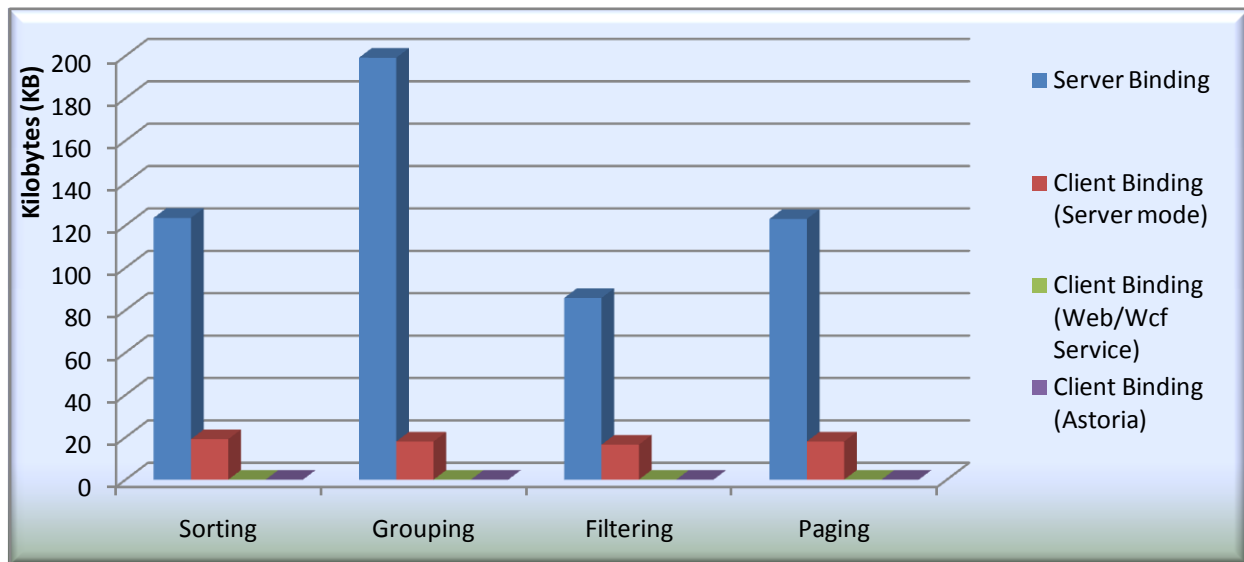
*The chart above compares several modes in various Grid scenarios. Lower graph is better, which means smaller response size.*

### Data footprint size comparison on transactional operation



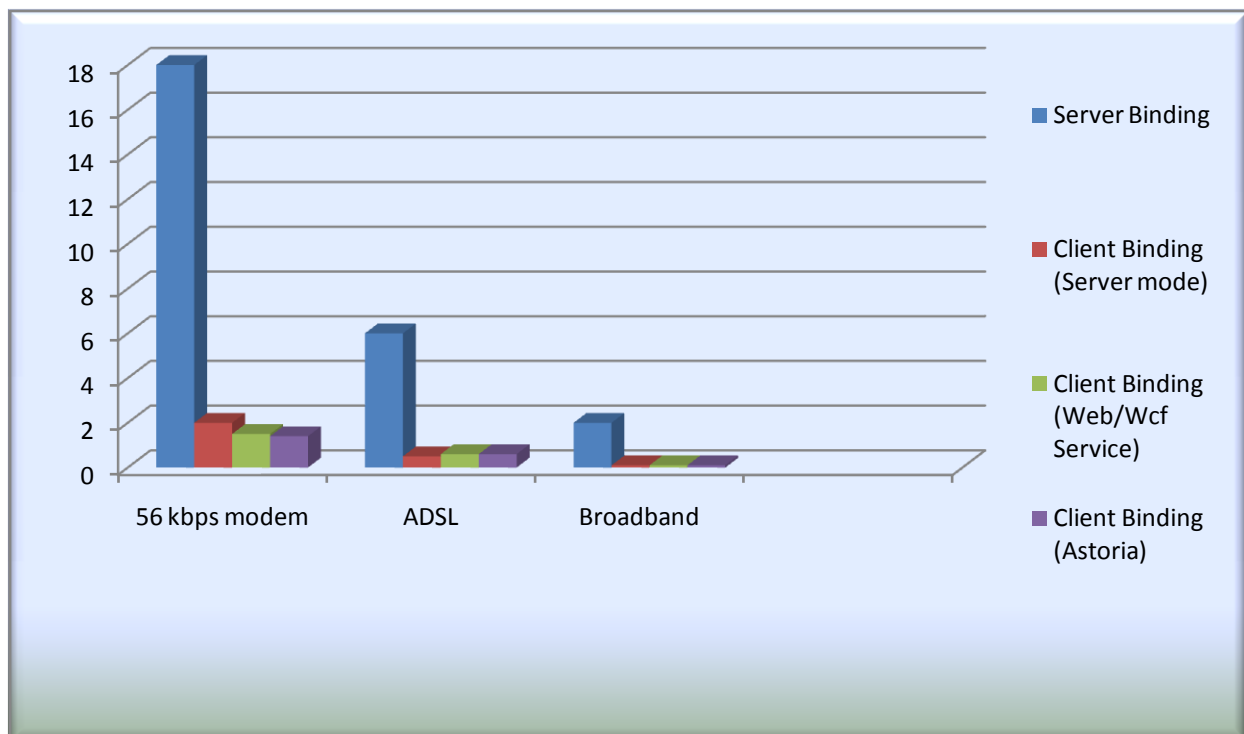
*Lower graph is better, which means smaller response size.*

### Data footprint size comparison based on various Grid functions and data load mode is set to “AllData”



*Lower graph is better. Notice that client binding with service mode have zero data footprint, since the data functions will be done entirely in client side.*

### Response time comparison on AJAX operation (based on data footprint in first illustration above)



*Lower graph is faster. The time measurement is per second.*

\*) Performance benchmark is tested on Intel® Core™ 2 Duo 2.4Ghz processor with 2 GB memory.

\*\*) Tested samples are using the combination of Northwind's Customers and Orders data table.



## Supported Features and Limitations

WebGrid Enterprise™ 7 features [unique client binding architecture](#) that addresses performance bottleneck as well as introduces numerous [benefits](#), without trading-off essential Grid features.

Intersoft's ClientBinding™ is designed to work in concert with key and fundamental features already implemented in WebGrid. The following features have been tested – and have been further enhanced to some extent – to fully support client binding operation mode:

- **Column and UI operations**

Column operations – such as resizing, moving and best fit – as well as UI operations – such as drag drop, context menu, filter bar etc – are fully supported. More advanced features such as column selection and column removing that require meticulous handling are supported as well.

Thanks to [VirtualRendering™](#), WebGrid's new technology that provides sophisticated state management as end user interacts with WebGrid, which enables WebGrid to perform dynamic row binding and rendering regardless of the UI state.

- **Layout features**

Most layout and appearance features are fully supported in client binding mode – including all details such as border, gridlines, and alternating row. As such, WebGrid renders high-quality, identical user interface in the client side, as if it were rendered from server-side.

- **Grid fundamentals**

WebGrid 7's ClientBinding™ is a comprehensive client binding architecture that takes account every fundamental functions of WebGrid, in addition to many new features and capabilities that it provides.

Fundamental WebGrid features – such as value list translation, column types, edit types, data formatting, custom editors and many of WebGrid's key features – continue to work flawlessly in client binding mode. To learn more on the framework enhancements that made it possible, please refer to [comprehensive client data features](#).

- **Sorting and Multiple Sorting**

Data sorting is fully supported in client binding mode. Multiple columns sorting with different sort direction is supported as well. For example, sort on ContactTitle in ascending order and OrderDate in descending order.

- **Grouping and Multiple Grouping**

Data grouping is entirely performed in the client side, which enables multiple grouping and grouping-related operations to be done without server-side callback. Several advanced grouping features are also supported in client binding, such as group caption formatting, various group intervals, and group mode.

- **Filtering and Multiple Filtering**

Client binding provides several implementation of filtering based on the selected data source type and selected loading mode. When using client service type of data source such as WebService and the service returns all data, WebGrid will perform filtering entirely in client side. However, if the service is configured to return paged data, WebGrid will fetch the data from the service through *DataSourceSelectArguments*. Refer to [client binding concept](#) to learn more.

- **Column Footer, Group Total and Aggregation**

Client binding fully supports data aggregation – such as Count, Average, Sum, Max and Min – which are fundamental features in a data grid. The aggregation operation is performed entirely in the client side without dependencies on server side, enabling WebGrid to recalculate the data aggregation anytime during user interaction. For instance, group total and column footer result will be updated instantly as user makes changes to the data.

- **Paging – VirtualLoad™ and Classic™**

Paging is a crucial feature in data grid and will continue to work flawlessly in WebGrid's new client binding mode. Regardless of the client datasource type, the paging function will work in consistent behavior as if it were operating in server binding mode. Depends on the data loading mode, WebGrid will automatically determine whether it should perform paging entirely in the client side, or request a paged data from your service.

- **Custom Tag Serialization (New)**

When operating in client binding mode, WebGrid sent raw data instead of markup to the client side. Often times, you will need to send custom information along with the data row being sent to the client. A new property *SerializeTagsToClient* is introduced to address this requirement, which is only applicable to *ServerDataSource* type.

- **Transaction operations**

Data transactions – such as insert, update and delete – work out-of-the-box in client binding mode. When *ServerDataSource* is used and WebGrid is bound to updatable datasource control, the transaction operations can be done automatically without additional codes. The same is true for *AdoDataSource* which has full capability in data operations. For more information on how to configure transaction operations in client service, please see [Transaction Operations \(Insert, Update and Delete\)](#)

- **Batch Editing (New)**

[SmartBatchUpdate™](#), a new major feature introduced in WebGrid Enterprise™ 7, is extensively designed to work perfectly in client-side binding mode. When used together with client binding feature, SmartBatchUpdate™ delivers a new editing experience that allows end user to view and make changes to data entirely in client side. The combination of these features, along with inline-editing transforms your application into offline-capable and cloud-ready Web.

- **LiveFreeze™**

Column freezing is supported in the same way it was supported in server-side mode. It has also been enhanced for client binding mode, so that it preserves consistent behavior every time the view has changed, such as upon sorting, filtering, paging – regardless of the client binding datasource type and loading mode.

- **Pivot Charting**

Since Charting require WebGrid to perform extensive data aggregation in server-side, Pivot Charting is supported only in [ServerDataSource](#) mode.

- **More features**

Intersoft's ClientBinding™ technology isn't merely a feature that capable to display plain data – or leaves advanced features disabled. It is rigorously engineered from scratch to ensure it covers advanced features available in WebGrid such as those explained above, as well as high extensibility to support upcoming features.

In fact, ClientBinding™ supports nearly all WebGrid features that could end up in hundreds. Some noteworthy features are preview row, columnset layout, automatic column fit, multiple selection, section 508 standards, automatic row restoration, input masking, custom editors and more.

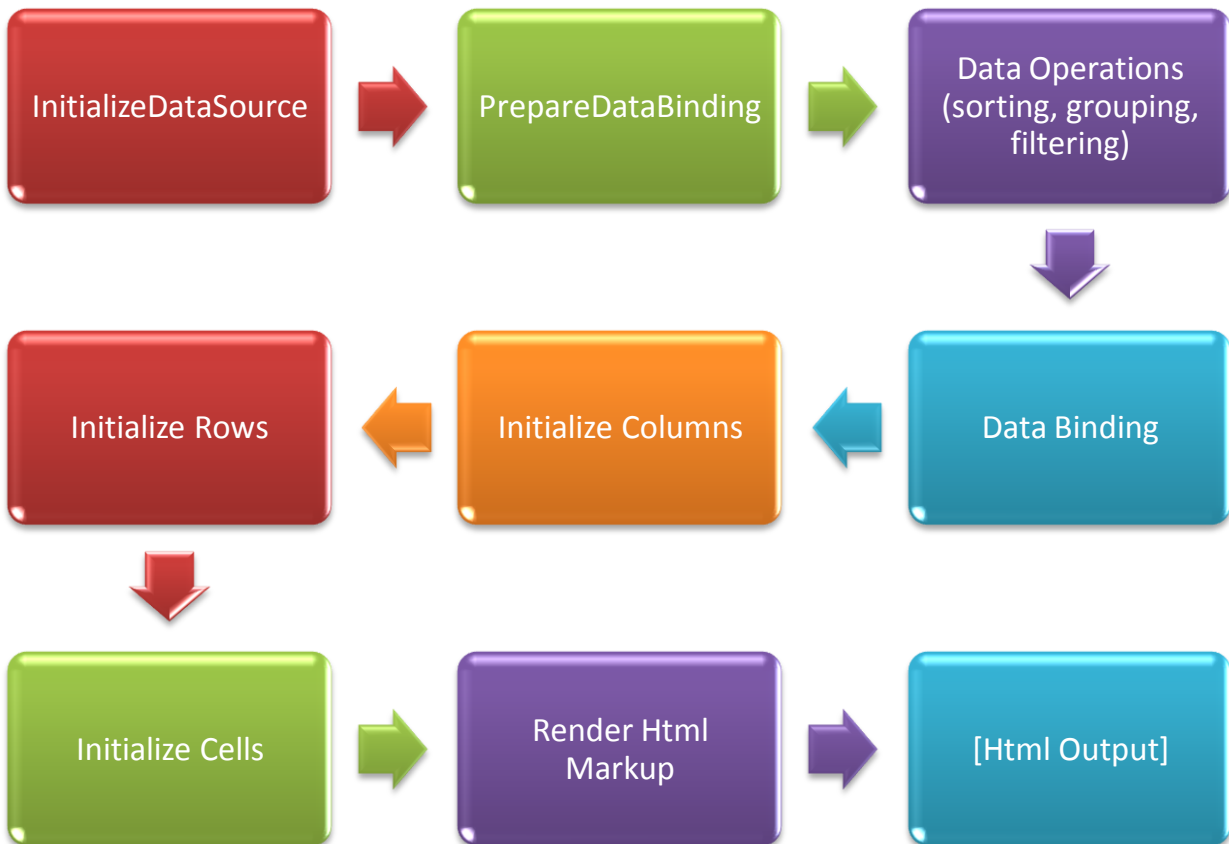
WebGrid Enterprise™ 7 is focused on complete support for client binding in flat Grid mode, to ensure high reliability with existing APIs and feature and best performance in various scenarios. The following features are not supported in version 7, although the architecture has been properly designed to support implementation for those features in the next version:

- Hierarchical Grid (Child Tables)
- Self Referencing

### The Differences with Server-side Binding

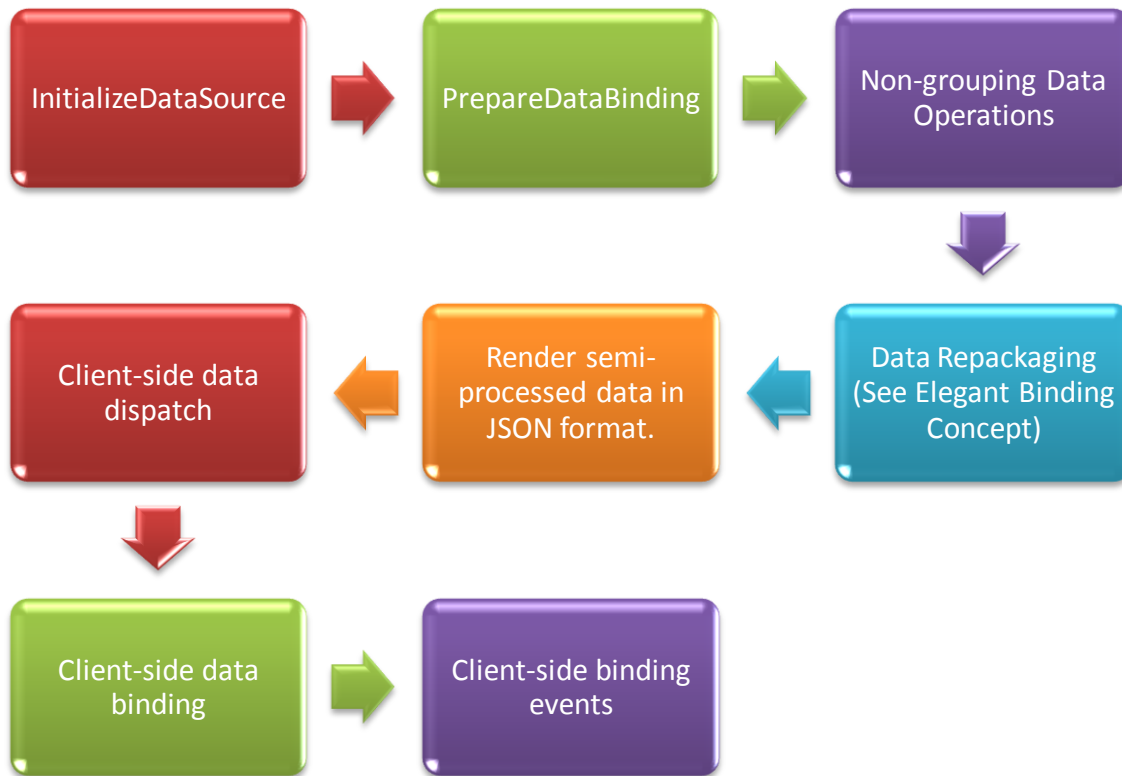
This section provides information on how client-side binding differs to server-side binding in terms of data processing and event life cycle. The details will also help you in upgrading existing implementation that used server-side binding into client-side binding properly.

Server-side binding processes data binding, initialization and rendering in the following way:



Client-side binding includes a special *ServerDataSource* mode to enable you to quickly leverage existing server-side infrastructure while taking advantage of [many benefits](#) introduced by client binding. This means that you can still connect WebGrid to a server-side data source – such as *SqlDataSource*, *ObjectDataSource*, *LinqDataSource* – or data source object assigned in *InitializeDataSource* event; while performing data binding operation in the client side.

Client-side binding using ServerDataSource mode processes data binding, initialization and rendering in the following way:



As you can see in the illustrated diagram above, client-side binding mode stripped several processes from the data binding, initialization and rendering process in server-side.

The key points of the differences can be summarized in the following:

- Physical data connection such as retrieving data from data source control and prepare data binding event will still be invoked in client binding mode. The life cycle is consistent in both first page load and FlyPostBack actions, and works in the same way as in server-side binding mode.
- Client binding shapes the raw data into semi-processed data, such as including important data such as value list translation, sorted rows, and more. It doesn't perform grouping operation.
- WebGrid intelligently repackages the data into rowset required only by the current view – taking account paging, filtering and other aspects.
- WebGrid sent the repackaged data in JSON format – [reducing over 90 percent of data footprint](#).
- Since the actual rendering does no longer occurring in the server side, the initialize events – such as InitializeTable, InitializeColumn, InitializeRow and InitializeCell – are no longer invoked.
- As data binding is performed in the client side, WebGrid 7 provides new client side events for data post-processing requirements.

As such, if you have existing codes that perform initialization in one of the above mentioned events, the codes will no longer work. You'll need to migrate the codes to the appropriate client-side events.

For example, consider the following C# codes to handle post-processing in InitializeRow server-side event:

```
protected void WebGrid1_InitializeRow(object sender, RowEventArgs e)
{
    if (e.Row.Type == RowType.Record)
    {
        WebGridCell hasAttachment =
            e.Row.Cells.GetNamedItem("HasAttachment");
        WebGridCell isUnread = e.Row.Cells.GetNamedItem("IsUnread");
        WebGridCell priority = e.Row.Cells.GetNamedItem("Priority");
        WebGridCell size = e.Row.Cells.GetNamedItem("Size");

        if (hasAttachment.Value.ToString() == "True")
            hasAttachment.Text = "./images/Attachment.gif";
        else
            hasAttachment.Text = "./images/wg_blank.gif";

        if (isUnread.Value.ToString() == "True")
            isUnread.Text = "./images/message.gif";
        else
            isUnread.Text = "./images/wg_blank.gif";

        if (priority.Value.ToString() == "1")
            priority.Text = "./images/high.gif";
        else
            priority.Text = "./images/wg_blank.gif";

        size.Text += " KB";
    }
}
```

Since InitializeRow is no longer invoked, the above codes will not be executed. To achieve the same post-processing in client binding mode, handle the *OnInitializeRow* client-side event. Thanks to Intersoft's client framework architecture that mimics server-side object model, you can easily convert the C# codes to Javascript such as shown in the following:

```
function WebGrid1_OnInitializeRow(id, row)
{
    if (row.Type == "Record")
    {
        var hasAttachment = row.Cells.GetNamedItem("HasAttachment");
        var isUnread = row.Cells.GetNamedItem("IsUnread");
        var priority = row.Cells.GetNamedItem("Priority");
        var size = row.Cells.GetNamedItem("Size");

        if (hasAttachment.Value.toString() == "true")
            hasAttachment.Text = "./images/Attachment.gif";
        else
            hasAttachment.Text = "./images/wg_blank.gif";

        if (isUnread.Value.toString() == "true")
            isUnread.Text = "./images/message.gif";
        else
            isUnread.Text = "./images/wg_blank.gif";

        if (priority.Value.toString() == "1")
            priority.Text = "./images/high.gif";
        else
            priority.Text = "./images/wg_blank.gif";
    }
}
```

```

        size.Text += " KB";
    }
}

```

With above codes, you can achieve custom requirements with identical results between client and server binding mode.

## Elegant Client Binding Architecture

WebGrid Enterprise™ 7 relies heavily on a solid client data framework in order to perform client binding efficiently, and to support existing features in various configurations and scenarios.

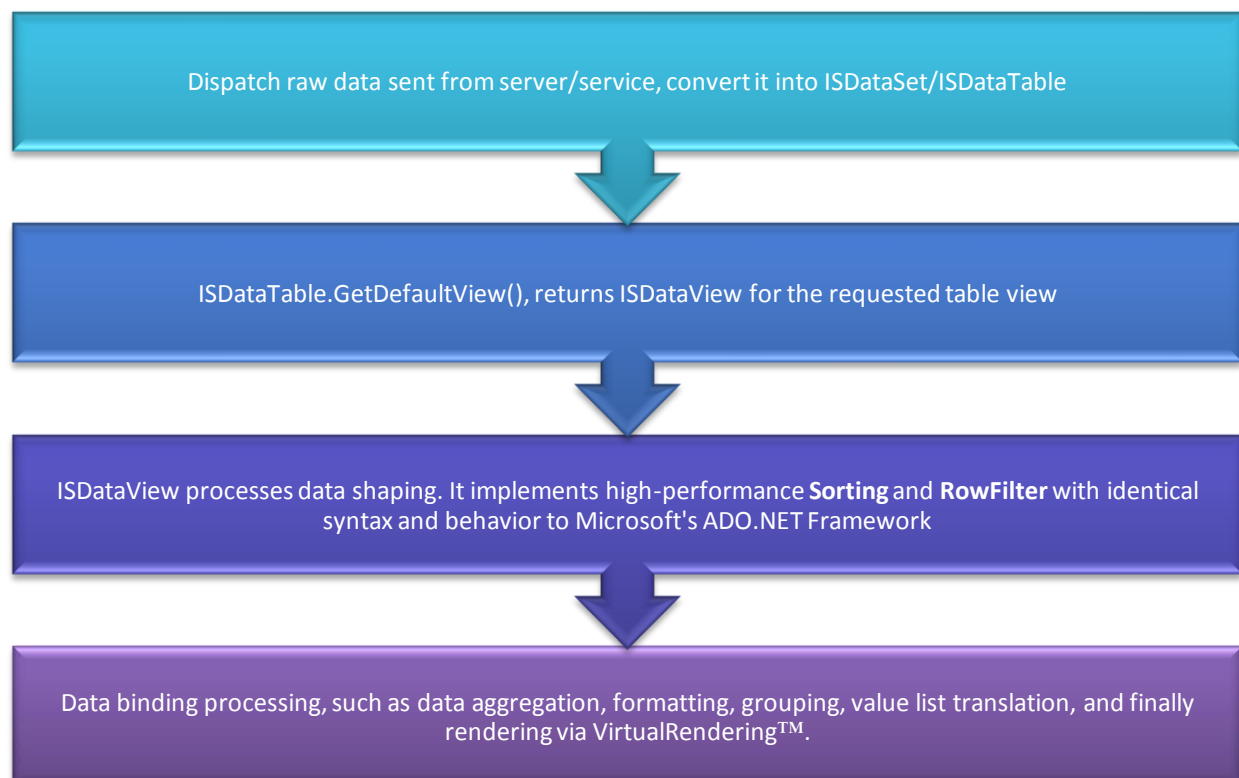
The following sections discuss Intersoft's client binding technology such as CDOF (Client Data Object Framework), data loading mode, transaction and batch update support, and more.

### Client Data Object Framework

Intersoft's Client Data Object Framework can be illustrated as a lightweight, mini version of Microsoft's ADO.NET Framework. *Client Data Object Framework* will be further called *CDOF* in the following section.

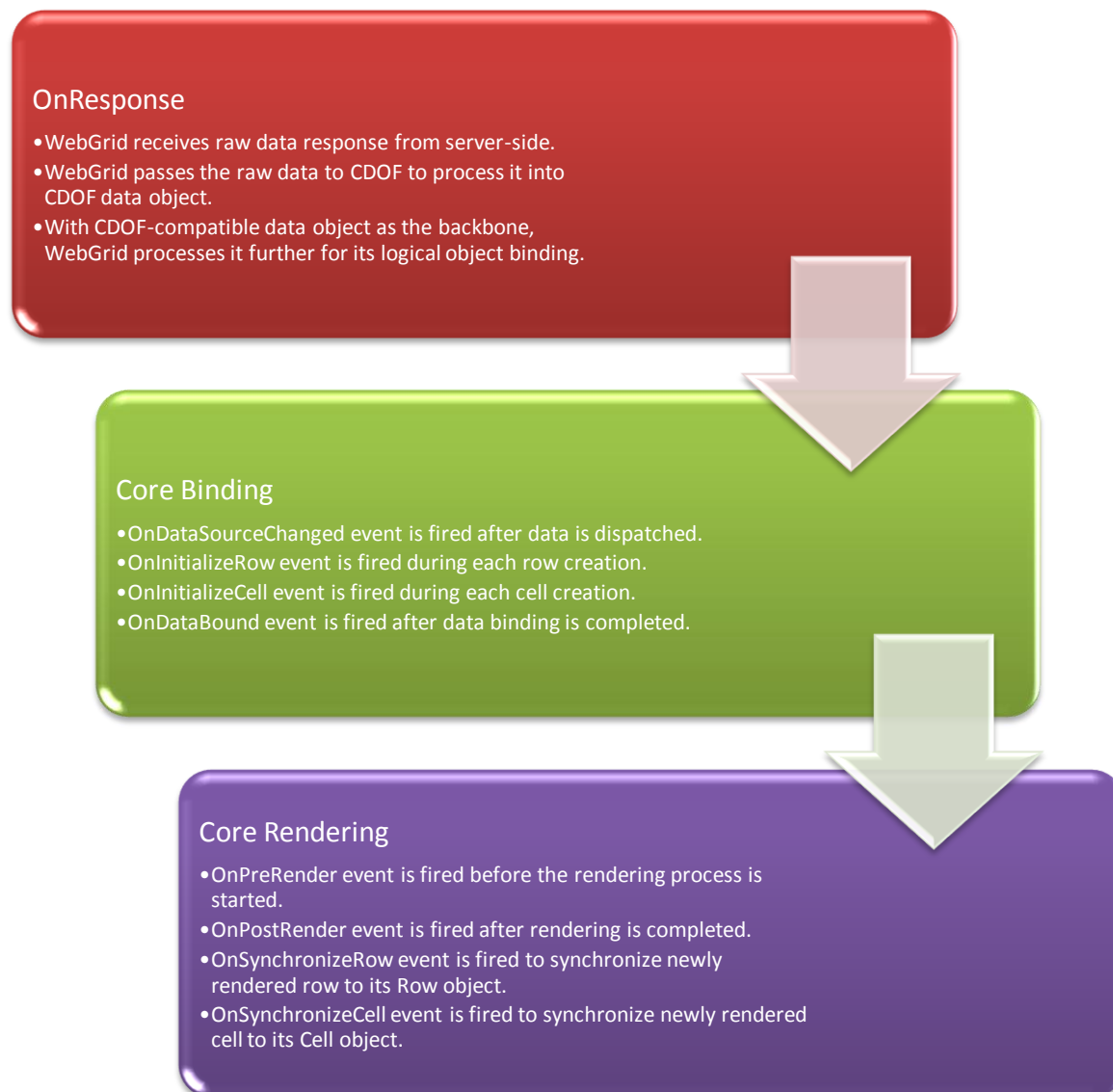
CDOF includes client-side implementation of DataSet, DataTable and DataView – which acts as the backbone of all client-side data operations in WebGrid. The *ISDataSet* and *ISDataTable* represent the main data object which is then used by WebGrid for further processing such as data shaping, formatting, paging and more.

The following illustration describes the overall client binding processing in WebGrid.



As shown in above illustration, WebGrid doesn't process data shaping by its own – but it relies on CDOF to obtain finalized data shape, which is then processed further by [VirtualRendering™](#). This unique architecture model enables clean separation between data abstraction layer and rendering layer – making it easy to be consumed by WebGrid and other WebUI Studio® family that requires client binding implementation in the future.

In addition to the overall binding process as shown above, it's also important to understand the new process model and events life cycle in WebGrid. The following information explains the client binding process life cycle in details:



With rock-solid foundation implemented in *Client Data Object Framework*, WebGrid Enterprise™ 7 sets a new standard for “cloud”-ready application by providing comprehensive, reliable client binding platform that supports enterprise scenarios and advanced features at the same time, as well as high extensibility for future’s platform and technologies.



## Rich Client-side Data Processing

In addition to providing solid client binding platform, Client Data Object Framework™ (CDOF) also enables many client-side data processing which were previously difficult or not possible to be achieved.

The following lists the key features of client-side data processing:

- **Client-side Data Formatting.**

One of the most powerful features in CDOF is its comprehensive data formatting function. It supports composite data formatting such as string, date time and number formatting. It also takes account Culture information. You can think it as a client-side implementation of .NET Data Formatting Library.

The data formatting feature is heavily used in client-side binding operation to display formatted values instead of raw values.

For example, your data service often returns number value in its simple form such as “50.9”. However, you may want to have WebGrid displayed the value using “\$ #,##.##0” format in “en-US” culture which will format the value as “\$ 50.90”.

WebGrid automatically honors the *DataFormatString* value that you specified in *WebGridColumn*, and will format it according to the selected culture info.

- **Client-side Sorting.**

CDOF implements high-performance client-side sorting that is capable to sort multiple columns in a fraction of seconds.

The client sorting also supports different sorting direction on each column. For instances, sort on Country descending, while ContactTitle ascending.

The client sorting is implemented at data access level instead of UI level – enables developers to have greater control over data shaping process. The sorting feature is implemented at *ISDataView* class.

- **Client-side Filtering.**

CDOF features comprehensive filtering support enabling developers to easily perform row filtering at data object level. The client-side filtering implements syntax that is fully compatible with server-side ADO.NET Framework’s filter expression – making it easy for .NET developers to consume client-side filtering features.

The client-side filtering supports the following standard operators:

- Equals to (=)
- Not equals to (<>)

- Greater than or equals to ( $\geq$ )
- Greater than ( $>$ )
- Less than or equals to ( $\leq$ )
- Less than ( $<$ )

It also supports more advanced operators such as:

- Like
- Contains
- Nested Conditioning Group

The value of the filter expression can be either one of the following types:

- **String.** The value part of the expression should to be enclosed by *quote* character (')
- **Boolean.** The value part can be written directly as it is.
- **Number.** The value part can be written directly as it is.
- **DateTime.** The value part should be enclosed by *sharp* character (#)

The following samples show various filter expression supported in CDOF.

*a. Basic string-type filter expression.*

`[ContactTitle] = 'Owner' and [Country] = 'USA'`

*b. Filter expression on various types with group.*

`(([ContactTitle] = 'Owner' or [ContactTitle] like 'sales') and ([OrderDate] > #1/1/2008# and [OrderDate] < #1/1/2009#))`

*c. Number filter expression.*

`[OrderAmount] > 123.45 and [OrderAmount] < 456.78`

*d. Filtering out null or empty data.*

`[Region] <> null and [Region] <> ''`

- **Client-side Paging.**

Because CDOF implements data shaping processing entirely in client side, it enables paging to be performed in client-side as well.

Client-side paging is fairly easy to be implemented at control/UI level, as long as final data shape has been made available. WebGrid 7 implements client-side paging in its own process to support more comprehensive paging options such as VirtualLoad™ and Classic™ paging, as well as to gain better control over the paging process based on various data source types and scenarios.

## Data Loading Mode

WebGrid 7's ClientBinding™ includes two type of data loading mode for client-based data service, regardless of the paging mode of WebGrid in User Interface level.

Data selection can be configured as easy as declarative property set – without requiring you to write Javascript code. For data selection to work, simply set the *SelectMethod* property in the *ServiceMethods* to the method name used for data retrieval in your data service.

Data loading modes are:

- **AllData.**

If your data service is designed to return all data, then you should choose *AllData* as the value of *DataLoadMode* property.

Tips: You can still enable UI paging in WebGrid level, such as VirtualLoad™ or Classic™ paging. When UI paging is enabled in this mode, WebGrid will perform paging in client-side. This means that page navigation will not require server round-trip, as it's performed and rendered entirely in client side. The same goes true for other data operations, such as sorting, filtering and grouping.

- **PagedData.**

If your data table contains relatively large amount of data, you may want to retrieve only subset of the data for the current view. In this case, you should consider using *PagedData* mode.

When using this mode, you are responsible to design your data service to return only paged data based on the information requested by WebGrid.

Intersoft's ClientBinding™ introduces an elegant approach that enables you to easily retrieve paged data in your data service. ClientBinding encapsulates essential select arguments into an object called *DataSourceSelectArguments*. This object is always passed to the parameter of your Select method, enabling developers to easily perform data selection based on the information. See [Paged Data C# sample](#) to see how ClientBinding™ handles data paging elegantly.

When data operation – such as sorting and filtering – is performed in *PagedData* mode, WebGrid will send a request to the specified data service by including complete request data in the *selectArguments* parameter. Developers are responsible to handle the sorting, filtering and paging based on the select arguments. In addition, WebGrid will also send a *SelectCount* request when it needs to invalidate the paging status.

**Important:** When *PagedData* data loading mode is used, you are required to enable one of the paging mode available in WebGrid, such as VirtualLoad™ or Classic™ paging mode.

*DataSourceSelectArguments* class contains the following properties:

- *FilterExpression*. The string of filter expression based on ADO.NET syntax.
- *SortExpression*. The string of sort expression based on ADO.NET syntax.
- *MaximumRows*. The number of maximum rows to be retrieved.
- *OperationType*. The type of the operation for this request.
- *StartRowIndex*. The start index of the row to be retrieved.
- *ViewName*. The view name or table name to be selected.
- *Tag*. Custom information passed from client-side.

Additionally, a special method *GetLinqFilterExpression* is provided to convert ADO.NET filter expression into LINQ-compatible filter expression. This method is specifically useful for data service that used LINQ to retrieve data.

The following C# sample shows how you can easily retrieve paged Orders data by using DataContext, LINQ to SQL, and dynamic LINQ.

```
[WebMethod]
public object GetPagedData(DataSourceSelectArguments selectArguments)
{
    NorthwindDataContext context = new NorthwindDataContext();
    context.DeferredLoadingEnabled = false;

    var pagedData = context.Customers.AsQueryable();

    // handle sorting
    if (!string.IsNullOrEmpty(selectArguments.SortExpression))
        pagedData = pagedData.OrderBy(selectArguments.SortExpression);

    // handle filtering
    if (!string.IsNullOrEmpty(selectArguments.FilterExpression))
        pagedData =
            pagedData.Where(selectArguments.GetLinqFilterExpression());

    if (selectArguments.OperationType == SelectOperation.SelectData)
    {
        // handle paging
        if (selectArguments.MaximumRows > 0)
            pagedData =
                pagedData.Skip(selectArguments.StartRowIndex).Take(
                    selectArguments.MaximumRows - selectArguments.StartRowIndex);

        return pagedData.ToList();
    }
    else if (selectArguments.OperationType == SelectOperation.SelectCount)
        return pagedData.Count();

    throw new InvalidOperationException("Unsupported operation type!");
}
```

## Transaction Operations (Insert, Update and Delete)

ClientBinding™ fully supports transaction operations for data service types – in the same elegant way as in data selection process.

The *ServiceMethods* object includes the following properties to handle transaction operations:

- InsertMethod.
- UpdateMethod.
- DeleteMethod.

Similar to SelectMethod, you need to have corresponding methods in your data service which is used to handle each operation. The method name is then set to each property.

ClientBinding™ handles the complex serialization and deserialization process automatically. As a result, you can develop your transaction methods in an elegant way by accepting the original and new item object. For this to work, you need to set the corresponding object name in *ItemType* property, and provide the class structure of your object in the client-side.

The following example shows how you can support update operation for Customers table in WebGrid.

First step, create the corresponding class of *Customer* in the client-side and set *ItemType* property to *Customer*.

```
function Customer()
{
    this.__type = "Customer";
    this.CustomerID = "";
    this.CompanyName = "";
    this.ContactName = "";
    this.ContactTitle = "";
    this.Address = "";
    this.City = "";
    this.Region = "";
    this.PostalCode = "";
    this.Country = "";
    this.Phone = "";
    this.Fax = "";
}
```

Next, create a method in your data service to handle the update operation. This sample is using WebService as the data service and LINQ-to-SQL as the data access.

```
[WebMethod]
public TransactionResult UpdateCustomer(Customer newObject, Customer
                                     originalObject)
{
    TransactionResult result = new TransactionResult();
    NorthwindDataContext context = new NorthwindDataContext();

    context.Customers.Attach(newObject, originalObject);

    try
    {
        context.SubmitChanges();
    }
}
```

```

    }
    catch (ChangeConflictException conflictException)
    {
        result.Exception = new JsonException(conflictException);
    }
    catch (Exception exception)
    {
        result.Exception = new JsonException(exception);

        /* set ExceptionHandled to true to suppress error message in the client
           side
           result.ExceptionHandled = true;
        */
    }

    result.OperationType = DataSourceOperation.Update;
    return result;
}

```

Finally, set the *UpdateMethod* of your WebGrid to *UpdateCustomer*.

As illustrated in the above sample, the update method consisted of the following processes:

- The method signature should return *TransactionResult* object and accept two parameters which are *newObject* and *originalObject*.
- If an error occurred during transaction process, assign the exception into *Exception* property. The type should be *JsonException* which encapsulates .NET Framework's original Exception object.
- Set the *OperationType* of the transaction result accordingly.
- Returns the transaction result.

Similarly, the Insert and Delete operation also includes patterned method signature. The following explains the method signature of each operation.

- public **TransactionResult** <InsertMethod> (<Object> newObject)
- public **TransactionResult** <UpdateMethod> (<Object> newObject, <Object> originalObject)
- public **TransactionResult** <DeleteMethod> (<Object> originalObject)

WebGrid 7's transaction operations work best with .NET Framework 3.5 data access technologies such as LINQ to SQL and ADO.NET Entity Framework. As the above sample shows, transaction operations can be done in elegant way, hassles-free and very straightforward.

## Batch Update Support

As if ClientBinding™ isn't [comprehensive](#) enough, it also includes a unique implementation of Batch Update that is provider independent – making ClientBinding™ the most advanced and reliable technology to address your ever-dynamic, “cloud”-ready enterprise Web 2.0 application.

WebGrid introduces SmartBatchUpdate™, a major feature introduced in version 7 that [revolutionizes data editing](#) by enabling multiple editing in the client-side and then submit all changes in a single request. This new batch update feature also supports batch update operation in data service.

Similar to data transaction operations, implementing batch update in data service can be done elegantly through strongly-typed object model.

The batch update method has the following signature:

```
public TransactionResult <BatchUpdateMethod> (List<ClientRowChanges> changes)
```

The *BatchUpdateMethod* property is also made available in *ServiceMethods* object, which you can specify according to the name of web method in your data service.

When batch update feature is enabled and the client binding mode is using data service, WebGrid will automatically submit all changes by invoking the web method specified in *BatchUpdateMethod*.

WebGrid submit all changes in an object collection of List<ClientRowChanges>. In your data service end, you can easily loop on the provided changes collection, and perform physical update according to the modification state of each change.

The following C# sample shows how to perform batch update in WebService. The sample used LINQ-to-SQL and DataContext as the data access.

```
[WebMethod]
public TransactionResult UpdateCustomers(List<ClientRowChanges> changes)
{
    TransactionResult result = new TransactionResult();
    NorthwindDataContext context = new NorthwindDataContext();

    foreach (ClientRowChanges change in changes)
    {
        try
        {
            if (change.RowState == ClientRowState.Added)
            {
                context.Customers.InsertOnSubmit((Customer) change.NewObject);
            }
            else if (change.RowState == ClientRowState.Deleted)
            {
                context.Customers.Attach((Customer) change.OriginalObject);
                context.Customers.DeleteOnSubmit((Customer) change.OriginalObject);
            }
            else if (change.RowState == ClientRowState.Modified)
            {
                context.Customers.Attach((Customer) change.NewObject,
                (Customer) change.OriginalObject);
            }
        }
    }
}
```

```

        catch (Exception exception)
        {
            result.Exception = new JsonException(exception);

            /* set ExceptionHandled to true to suppress error message in the
client side
            result.ExceptionHandled = true;
            */
        }
    }

    // no errors during object attachment, ok to proceed
    if (result.Exception == null)
    {
        try
        {
            // submit all changes in batch
            context.SubmitChanges();
        }
        catch (Exception exception)
        {
            result.Exception = new JsonException(exception);
        }
    }

    result.OperationType = DataSourceOperation.BatchUpdate;
    return result;
}

```

The batch update method works in similar way to other data operations, except it includes more information about a row change which is encapsulated in ClientRowChanges object.

The ClientRowChanges consisted of the following properties:

- RowState. *The changes state of the row.*
- TableName. *The table name of the changed row.*
- NewObject. *The new object that user adds in the client side.*
- OriginalObject. *The original object that doesn't contain user changes, which is required for Update and Delete operation.*

### Service Events

All data operations in data service are powered with a solid, provider-based architecture that is highly extensible and scalable.

Client data services that supported by WebGrid 7 such as *WebService*, *WcfService* and *AstoriaDataSource* inherit the same data source provider base. As a result, these data services share the same process life cycle, same events and behaviors.

Client data services support the following events:

- Selecting. Invoked when the data provider is about to perform data selection.
- Selected. Invoked after the data provider has successfully selecting data.
- Updating. Invoked when the data provider is about to perform data update.



- Updated. Invoked after the data provider has successfully updating data.
- Inserting. Invoked when the data provider is about to perform data insert.
- Inserted. Invoked after the data provider has successfully inserting data.
- Deleting. Invoked when the data provider is about to perform data delete.
- Deleted. Invoked after the data provider has successfully deleting data.
- BatchUpdating. Invoked when the data provider is about to perform batch update.
- BatchUpdated. Invoked after the data provider has successfully perform batch update.

Each operation includes pre and post event for complete customizability. The pre event such as *Selecting* can be cancelled with a return false statement.

These service events are available in *ServiceEvents* object which is located in *ClientBindingSettings*.

## Client Data Source and Data Service

The following sections explain the client data source implemented in ClientBinding™.

There are three general types of client data source:

1. [Server-side data source](#).
2. Service-based data source.  
ClientBinding™ includes three built-in data providers implementation for data service, which are [Web Service](#), [WCF Service](#) and [ADO Data Service](#).
3. Client-side data source.

### Server-side data source

This data source type enables you to continue using server-side data source in WebGrid. Supported server-side data source are:

- Datasource controls, such as *AccessDataSource*, *LinqDataSource*, *ObjectDataSource*, etc.
- Datasource object, which is assigned in *InitializeDataSource* event (also known as Traditional Binding).

This data source type is designed to help you leverage and reuse existing infrastructure which has used server-side data source, while at the same time enables you to take advantage of many [client binding benefits](#) such as improved performance and reduced data footprint.

To learn the difference between server-side binding mode and client-side binding mode with server-side datasource type, please see [The Difference with Server-side Binding](#).

Server-side data source type is the quickest and easiest way to take advantage of client-side binding. You don't need to create web service or data service as WebGrid seamlessly connects to server-side data source for data retrieval, which is then repackaged into lightweight format for further processing in the client-side.

## Web Service

WebService data source type enables you to connect WebGrid to a web service in elegant fashion, which is done through properties configuration without requiring any Javascript codes.

To connect WebGrid to a web service, simply specifies the service address in the provided *ServiceUrl* property and specifies the web method used to retrieve data in *SelectMethod* property.

WebService will automatically activate [full client-side operation](#) mode when your data service returns all data. It also supports [paged data retrieval](#) for optimized performance.

WebService fully supports [data transaction operations](#) such as insert, update, and delete. It also supports more advanced operation such as [batch update](#).

*Markup example:*

```
<ClientBindingSettings DataSourceType="WebService" ServiceUrl="~/Service1.aspx">
    <ServiceMethods SelectMethod="GetData"/>
</ClientBindingSettings>
```

**Important:** Your web service should have capability to return output in JSON format as client binding will accept only data in JSON format. If you're developing web service in ASP.NET 3.5, then your web service has supported JSON response format.

## Windows Communication Foundation (WCF) Service

WcfService data source type enables you to connect WebGrid to a Windows Communication Foundation service in elegant fashion, which is done through properties configuration without requiring any Javascript codes.

To connect WebGrid to a WCF service, simply specifies the service address in the provided *ServiceUrl* property and specifies the web method used to retrieve data in *SelectMethod* property.

WcfService will automatically activate [full client-side operation](#) mode when your data service returns all data. It also supports [paged data retrieval](#) for optimized performance.

WcfService fully supports [data transaction operations](#) such as insert, update, and delete. It also supports more advanced operation such as [batch update](#).

*Markup example:*

```
<ClientBindingSettings DataSourceType="WcfService" ServiceUrl="~/WcfService.svc">
    <ServiceMethods SelectMethod="GetData"/>
</ClientBindingSettings>
```

**Important:** Your web service should have capability to return output in JSON format as client binding will accept only data in JSON format. If you're developing WCF service in ASP.NET 3.5, then your web service has supported JSON response format.

## ADO.NET Data Service (Astoria)

ADO.NET Data Service is Microsoft's latest data service technology introduced in .NET Framework 3.5 SP1. It enables data to be consumed programmatically through standard Web protocols such as REST, XML, SOAP and JSON. To learn more about ADO.NET Data Service, please visit [ADO.NET Data Service Home](#).

Intersoft's ClientBinding™ is the first in the industry that implement full support for this cutting-edge technology.

To connect WebGrid to an ADO.NET data service, all you need to do is specifying the service address in the provided *ServiceUrl* property.

Astoria enables programmatic data access over the Web with very minimal efforts, which includes support for sorting, filtering and paging. It also includes native support for data transactions such as insert, update, delete, as well as more advanced operation such as batch update.

ClientBinding™ fully takes advantage of Astoria capabilities and implement direct interface to access Astoria's functionalities. As a result, you are not required to specify any of the service methods.

The following example shows a simple declarative markup to enable all data operations.

```
<ClientBindingSettings DataSourceType="AdoDataService"
                        ServiceUrl="~/WebDataService1.svc" ItemTypeName="Customer">
</ClientBindingSettings>
```

**Note: The current version of Astoria doesn't support complete paging feature yet. Therefore, you can only use [AllData](#) loading mode when using AdoDataService type.**

## Client-side data source

Client data source type allows you to assign a completely custom data source in client-side. When this type is used, WebGrid doesn't perform data loading automatically as the required datasource is not available.

This type is specifically useful when you need to retrieve datasource from external sites or data services that are not supported in Intersoft's ClientBinding™. For example, you may have a requirement to fetch data from an external site manually in the client-side, and then pass the resulted data to WebGrid via API calls for data binding.

Pure client-side binding takes as little as three lines of code, such as shown in the following:

```
grid.SetDataSource(dataSource);
grid.DataBind();
grid.Render();
```

WebGrid accepts ISDataSet, ISDataTable or any array-based collection as data source. For more information about client binding API, please see [Client Binding API](#)

## Client Binding Consideration and Best Practices

This section discusses the consideration for using client binding mode in your application, along with best practices for several common scenarios.

### Considerations

Although client binding offers [numerous benefits](#), it doesn't always fit to all scenarios. If your application is running locally in corporate network or on fast broadband network, and that your application doesn't require connecting to a service-based datasource, you may consider to stay with server binding mode.

Likewise, if your target users run on thin-client with slightly limited resources, or slower terminal – client binding might not be suitable as client binding performs all operations – such as populating data, formatting, sorting, filtering, grouping, paging and more – entirely in client side.

Client binding can be generally viable with paged data loading mode, which balances the workload between server and client side. The following sections describe best practices for several scenarios.

### Best practice #1 – Using *ServerDataSource* with Paging enabled

If you're already using WebGrid in your application (which means they were using server binding by default), you may consider using [ServerDataSource](#) mode to reuse your existing server assets and infrastructure when migrating to client binding.

*ServerDataSource* is the mode that requires the most minimal changes to your code. If you don't have post-binding processing in your codes (such as codes in *InitializeRow*, *InitializeCell* etc), the migration to client binding is almost seamless without any changes required.

In the above case, you simply need to set *DataSourceType* of *ClientBindingSettings* to *ServerDataSource*. Run your page to ensure everything is working fine.

### Best practice #2 – Using *WebService* with *PagedData* loading mode

If you're considering developing application that retrieves data from data service, you can consider using [WebService](#) or [WcfService](#) depending on your server infrastructure.

To anticipate the growth of the data size returned by your data service, you can prepare your data service to return [paged data](#) instead of all data. This ensures your application to extend and scale up properly in the future.

### Best practice #3 – Using *AdoDataService*

If you're considering developing data service platform that is accessible over the Web from various clients such as Silverlight or Web application – you may consider building ADO.NET data service.

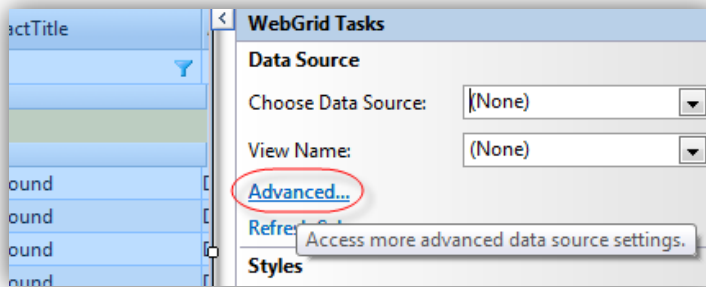
### Limitations

You can't use client-side binding when your WebGrid require specific features that are not supported in this version of WebGrid. For more information on the limitations, please see [unsupported features in client binding](#).

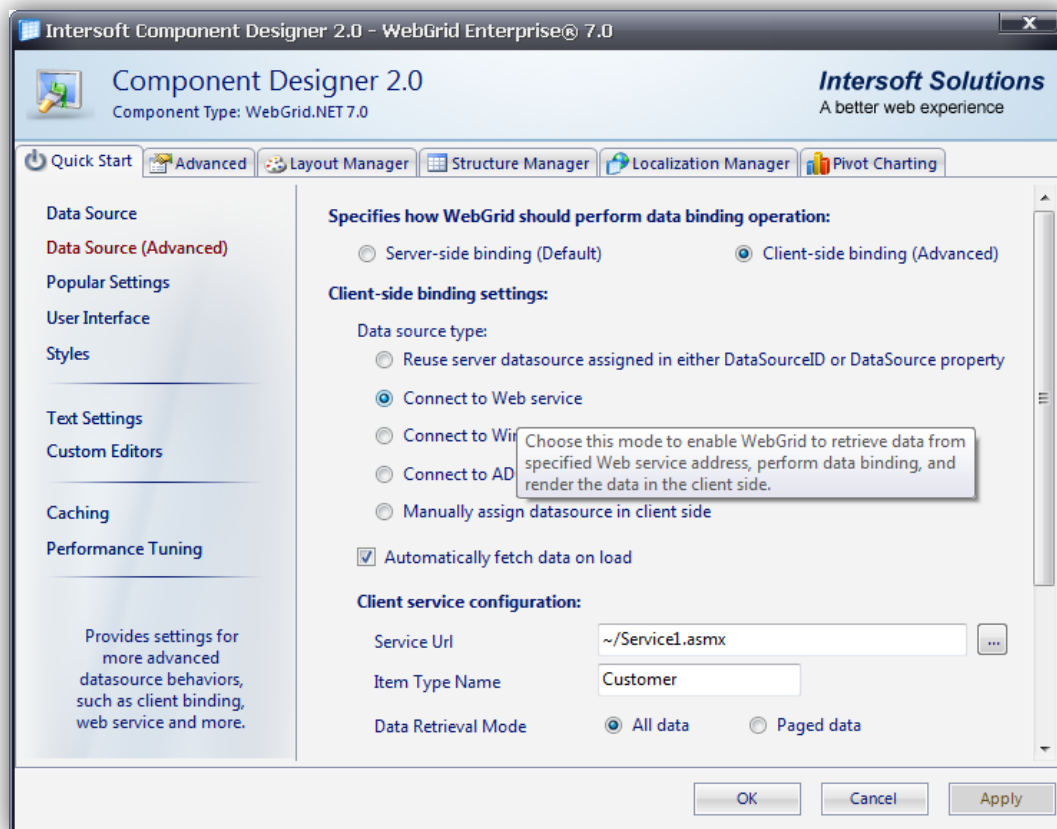
## Client Binding Configuration with Component Designer

ClientBinding™ provides comprehensive settings to support various scenarios used for enterprise Web 2.0 application. Thanks to the improved *Smart Tag* and *Component Designer* – making client binding configuration very easy.

To quickly getting started with client binding, open WebGrid's Smart Tag, then click **Advanced** as shown in the following.



WebGrid's award-winning **Component Designer** streamlines all client binding configurations in an intuitive user interface, making it easy for you to work with various client binding options. For your convenience, it also includes hints/tooltip description in each option.



## Client Binding API

The client binding architecture included in WebGrid Enterprise™ 7 is a set of comprehensive new classes, methods and interfaces to enable programmatic consumption in elegant fashion. Most of the user interface functions and high-level encapsulation of binding process called the same underlying APIs.

The following sections explain the new public methods and events which are available when client binding mode is enabled and the data source type is set to either client services or client data source.

### New methods

Several methods have been added as extensions to provide programmatic client binding functionality. These methods will only be available when client binding feature is enabled.

Class	New Method	Parameters
<b>WebGrid</b>	SetDataSource (Assign a new client-side datasource to WebGrid)	dataSource, tableName
	GetConvertedDataSource (Gets the datasource that has been converted to CDOF-compatible object)	-
	DataBind (Instructs WebGrid to perform data binding based on assigned datasource)	-
	LoadData (Instructs WebGrid to perform initial data loading)	-
	Render (Perform rendering based on bound data)	-
	RebindData (Rebind WebGrid with new datasource)	dataSource, skipDataDispatch
<b>WebGridTable</b>	GetUngroupedRows (Gets ungrouped rows collection of this table)	-
	GetRowByKeyValues (Gets the WebGridRow based on the specified key values)	keyValues
	DispatchDataTable (Gets the CDOF data table object)	-
	GetView (Gets the CDOF data view object)	-
	DataBind (Perform data binding for this table)	-
	FillDataSourceArguments (Populate view state into data source arguments used for data retrieval operation)	srguments, sortState, filterState, pagingState
	GetSortExpression (Gets the sort expression required to construct the view for this table)	
	GetFilterExpression (Gets the filter expression required to construct the view for this table)	
	Render (Render the view for this table)	subTable

	UpdateDataStatus (Updates the status of current datasource to UI)	-
<b>WebGridRow</b>	GetGroupRowLevel (Gets the group row level of this row)	-
	HasCollapsedParent (Whether this row has collapsed parents)	-
	HasGroupedChilds (Whether this row has grouped child rows)	-
	HasExpandedChilds (Whether this row has expanded child rows)	-
	ExpandGroupRowsToThisRow (Expand all parent group rows down to this row)	-
	CopyTo (Copies this row's structure and data to destination object)	targetObject, isModifiedOnly, followTargetStructure

### New client side events

The following table describes the new client side events related to client binding feature.

Event Name	Description	Parameters
<b>OnCustomAggregate</b>	Fired when WebGrid needs to perform custom aggregation for column which AggregateType is set to Custom.	controlId, columns, rows, type
<b>OnInitializeRow</b>	Fired when a row is initialized.	controlId, row
<b>OnInitializeCell</b>	Fired when a cell is initialized.	controlId, cell
<b>OnSynchronizeRow</b>	Fired when a row requires synchronization.	controlId, row
<b>OnSynchronizeCell</b>	Fired when a cell required synchronization.	controlId, cell
<b>OnDataSourceChanged</b>	Fired when data source has been changed.	controlId
<b>OnDataBound</b>	Fired when data binding process is completed.	controlId
<b>OnPreRender</b>	Fired before WebGrid enters rendering phase.	controlId
<b>OnPostRender</b>	Fired after rendering operation is completed.	controlId

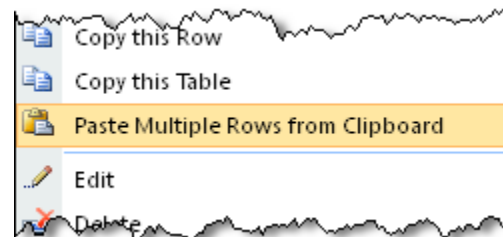
## Other Enhancements

In addition to major new features such as described in above topics, WebGrid Enterprise™ 7 also includes enhancements to various areas based on customer feedback. These enhancements are designed to help you achieving complex scenarios in professional approach.

### Improved user interface

WebGrid Enterprise 7 improves various user interface factors, making it more intuitive for end user to work with information. Many of these enhancements work in concert with new features introduced in WebGrid Enterprise™ 7.

For instance, when both batch update and allow add feature are enabled, a special “paste multiple rows” menu item will appear in the row context menu. This command enables you to import data from external documents such as Excel® spreadsheet – and then easily transfer the data into WebGrid with a single click.



WebGrid 7 also includes *adaptive status bar*, a new user interface enhancement that automatically arranges all commands in the status bar – making it hassle-free for you to use WebGrid features without have to worry your user’s screen real-estate.





## New client side events for inline editing

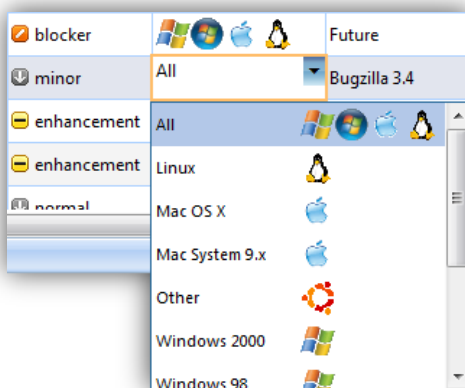
With strong focus in data editing enhancement, WebGrid 7 enables developers to have greater control and customizability over the editing process by introducing four new client side events as listed below:

- OnAfterExitEditMode***. The parameters are *controlId*, *tableName*, *editObject*.  
 This event is invoked after user exit cell-level editing. Unlike *OnExitEditMode* event, *OnAfterExitEditMode* is fired after default editing process is completed, making it possible for you to inspect the new cell values and perform custom logic.
- OnBeginRowEditing***. The parameters are *controlId*, *row*.  
 This event is invoked when user begin to edit a row. Unlike *OnEnterEditMode*, this event is fired only one-time when a row is about to be edited. Furthermore, this event is applicable on row-level instead of cell-level.
- OnEndRowEditing***. The parameters are *controlId*, *row*.  
 This event is invoked when user ended row editing completely. Similar to *OnBeginRowEditing*, this event is applied on row-level, and is called only one-time when user ended row editing. This event is fired after *OnExitEditMode* and before *OnRowValidate*.
- OnCancelRowEditing***. The parameters are *controlId*, *row*.  
 This event is invoked when user canceled row editing. This new client side event enables you to process custom logic when user canceled the editing. For example, you may want to validate if the cancelation has met specific condition before the cancelation is allowed to be processed further.

## Enhanced integration with WebCombo

Since its initial release, WebGrid Enterprise™ has been well known with its tight integration with other Intersoft's products, especially integration with WebCombo™.

In version 7, WebGrid enhances the WebCombo integration further in various scenarios such as when used in Custom and Image column type.



WebGrid 7 enables you to achieve more complex scenarios to deliver richer user experience, such as using Custom column type to display custom images. You can use WebCombo to allow your users to edit such type of information more intuitively.

When WebCombo is used in Custom column type, WebGrid automatically map the cell's value to the text field of WebCombo, while maintaining default inline editing behavior.

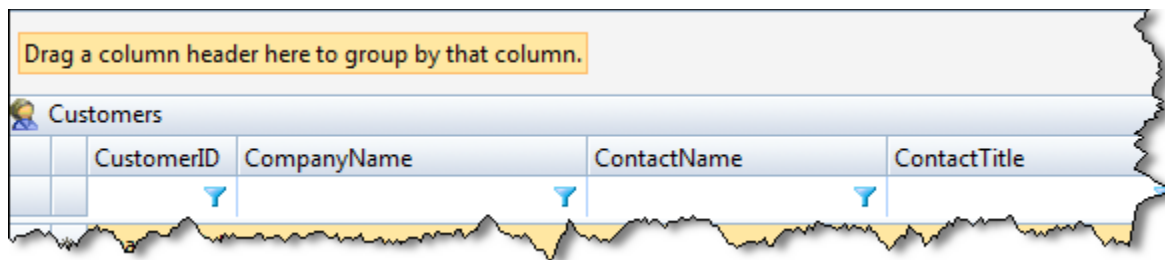
## Default Style Merging

WebGrid Enterprise™ 7 introduces *Default Style Merging*, a new feature that automatically merges your customized styles with default style.

Default Style, a feature introduced in version 5 is designed to let you easily change WebGrid's theme in a single property set. However, you can no longer customize WebGrid styles when you activate this feature without additional workaround. Default Style Merging is designed to address this limitation, makes it easy for you to customize WebGrid styles while preserving the original default styles.

To enable Default Style Merging feature, simply set *AllowDefaultStyleMerging* property to *true*. This new property can be found in *LayoutSettings* object.

For example, the following screenshot shows WebGrid with Elegant default style.

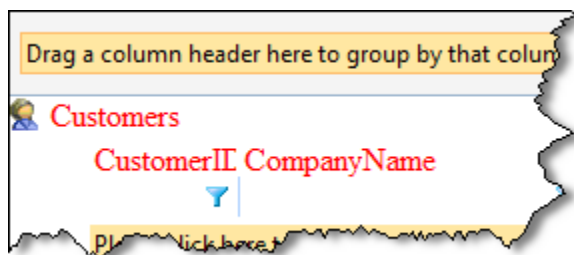


Next, assume you would like to change the font's color to Red, you would have the following style definition in *LayoutSettings*.

```
<HeaderStyle ForeColor="Red" />
```

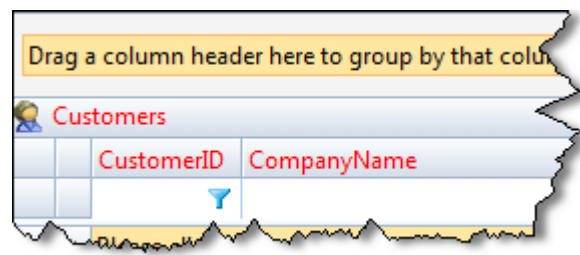
The following shows the comparison two different results.

### Without Default Style Merging



*The header's default style is lost after you defined custom header style.*

### With Default Style Merging



*The header's default style is flawlessly merged with your custom header style.*