TMS ASP.NET COMPONENT PACK V2.6

AdvWebGrid DEVELOPERS GUIDE

April 2010

Table of contents

AdvWebGrid availability

AdvWebGrid is part of the TMS ASP.NET Component Pack

The TMS ASP.NET Component Pack is available for:

**Microsoft™Visual Studio.NET 2003
Microsoft™Visual Studio.NET 2005
Microsoft™Visual Studio.NET 2005 with Microsoft Ajax
Microsoft™Visual Studio.NET 2008 with Microsoft Ajax
Microsoft™Visual Studio.NET 2010 with Microsoft Ajax**

**Current version of AdvWebGrid has been designed for and tested with IE6, IE7, IE8, Firefox 3.0, Firefox 3.5, Chrome 4.0
Some features, such as gradient support, are available only in IE6, IE7, IE8**

AdvWebGrid use

The TMS AdvWebGrid component is designed to be used in all kinds of grid type data presentation and editing in a browser. Data presented in the grid can be database driven as well as directly web application driven. It is from the web application running on the server that this grid presentation layer is generated along with JavaScript code that is executed in the browser on the client side. AdvWebGrid has built-in support for paged output, making the amount of data that is transferred from the server to the client customizable. AdvWebGrid can make use of cached script libraries or not. The property CachedScript controls this. It is set to false by default for easy deployment. To minimize server load and bandwidth, set CachedScript to true and make sure the files tms_advwebgrid.js and tms_griddatepicker.js are in the project folder.

AdvWebGrid organisation

The grid components consist of 4 parts:

1 : Controller

This is the part of the grid from where paging is controlled and presentation of page selection is done. Various options are available to customize the appearance of this control

2 : Header

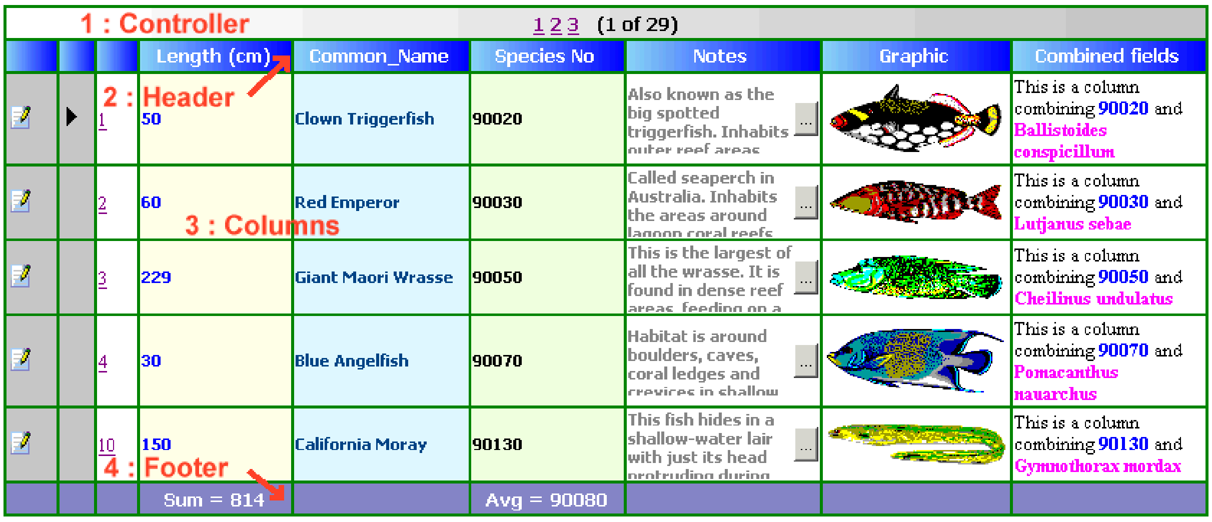
A one or two row header can be used. In its most simple form, the header indicates what data is displayed in each column. It can be used to trigger a column sort, to start a filter, to resize a column or to indicate logically grouped columns by spanning multiple columns.

3 : Columns

The actual data of the grid is displayed in columns. The appearance and type of each column can be defined.

4 : Footer

A footer can be used to display just text information, static column totals or dynamic column totals.



1 2 3 (1 of 29)

			Length (cm)	Common_Name	Species No	Notes	Graphic	Combined fields
✎	▶	1	50	Clown Triggerfish	90020	Also known as the big spotted triggerfish. Inhabits outer reef areas ...		This is a column combining 90020 and Ballistoides conspicillum
✎		2	60	Red Emperor	90030	Called seaperch in Australia. Inhabits the areas around lagoon coral reefs ...		This is a column combining 90030 and Lutjanus sebae
✎		3	229	Giant Maori Wrasse	90050	This is the largest of all the wrasse. It is found in dense reef areas, feeding on a ...		This is a column combining 90050 and Cheilinus undulatus
✎		4	30	Blue Angelfish	90070	Habitat is around boulders, caves, coral ledges and crevices in shallow ...		This is a column combining 90070 and Pomacanthus nauarchus
✎		10	150	California Moray	90130	This fish hides in a shallow-water lair with just its head protruding during ...		This is a column combining 90130 and Gymnothorax mordax
			Sum = 814		Avg = 90080			

Settings of AdvWebGrid visual elements

Controller

The controller part of the grid can perform the automatic paging of grid data. A page is equivalent to RowCount rows. For the DB-driven grid, the total number of pages will be the total number of rows in the dataset divided by RowCount. For the non DB driven grid, the total number of rows is set with the property TotalRows. The page controller then provides selected methods such as Previous page / Next page or direct page number hyperlinks to select the desired page. The look of the page controller is set through the grid's Controller property with following subproperties:

Controller properties:

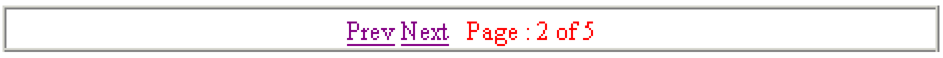
Alignment: sets the alignment of the text displayed in the controller.
BackColor: this sets the background color of the controller. Specify Color Empty if default browser background color should be used. Also specifies the start color of a gradient color in the controller. When Color Empty is set, no gradient is used. Note that gradients are only supported in IE6.
BackColorTo: specifies the end color of a gradient color in the controller.
Borders: Sets the border appearance of the controller
Caption: this is the text displayed along with the automatic displayed page control elements.
GradientOrientation: sets the gradient direction to either horizontal or vertical
Height: specifies the height of the controller. When height is zero, height of the controller automatically adapts to height of elements inside the controller.

HintFirst: sets the hint that should appear over the button to go to the first page
HintLast: sets the hint that should appear over the button to go to the last page
HintNext: sets the hint that should appear over the button to go to the next page
HintPrev: sets the hint that should appear over the button to go to the previous page
ImgFirst: specifies the image to be used for jumping to the first page. This can be a GIF, JPEG, PNG or BMP file. Note that the images are used when PagerType is set to Image
ImgLast: specifies the image to be used for jumping to the last page.
ImgNext: specifies the image to be used for jumping to the next page.
ImgPrev: specifies the image to be used for jumping to the previous page.
IndicatorFormat: This specifies how the viewed page is indicated in the controller. Text and number specifiers can be used. The numeric data displayed depends on the IndicatorType and can be either record number or page number.

Examples:

Setting IndicatorType tot PageNr and IndicatorFormat to '(Page {0} of {1})' will display in the controller: '(Page 1 of 12)' if on the first page for a 12 page grid.

Setting IndicatorType to RecordNr and IndicatorFormat to '- This is record number {0}' will display in the controller '- This is record number 1' for the first record.



Sample controller with prev / next hyperlinks and page indication

IndicatorType: Indicator type can be either RecordNr, to indicate current record index vs total number of records in the dataset or grid, or PageNr to indicate the current page. The IndicatorType None displays no indicator.
MaxPages: sets the maximum number of pages to display in the controller. The controller will display a range of pages limited to MaxPages around the currently selected page.
Pager: this selects the type of paging. Currently, following paging types are defined:
PageList: paging is done through clicks on page number
PrevNext: paging is done through previous / next links
PrevNextFirstLast: paging is done through previous / next page links as well as first and last page links.
DropDownList: paging is done trough selecting a page number from a dropdown-list

PagerType: this selects the visual presentation of the paging which can be:
Link: page numbers or previous / next actions are done through hyperlinks
Button: page numbers or previous / next actions are done through buttons
Image: previous, next, first and last actions are done through images specified in properties ImgPrev, ImgNext, ImgFirst, ImgLast. Note that PagerType Image cannot be combined with Pager PageList currently.

Position: sets the position of the controller w.r.t. the grid. This can be:
Top: controller is on top of the grid
None: controller is not displayed
Bottom: controller is displayed under the grid
Both: controller is displayed on top and under the grid

RowCountSelect : when true, a dropdown list is displayed in the controller with which the number of rows to display can be selected. The possible rowcount values are set through the RowCountValues property

RowCountValues: list of possible rowcount values that can be selected from a dropdown list in the controller.
TextFirst: sets the text for the button or hyperlink that will be used to go to the first page. By default this is 'First'
TextLast: sets the text for the button or hyperlink that will be used to go to the last page. By default this is 'Last'
TextNext: sets the text for the button or hyperlink that will be used to go to the next page. By default this is 'Next'
TextPrev: sets the text for the button or hyperlink that will be used to go to the previous page. By default this is 'Prev'

Events associated with the controller:

Although the controller handles the paging automatically, events are triggered to indicate to the application what paging action the user has taken. These events are:

GotoPage: event triggered when user selects to go to a given page when the controller pager is set to PageList

FirstPage: event triggered when user selects to go to the first page when the controller pager is set to PrevNext or PrevNextFirstLast

LastPage: event triggered when user selects to go to the last page when the controller pager is set to PrevNext or PrevNextFirstLast

NextPage: event triggered when user selects to go to the next page when the controller pager is set to PrevNext or PrevNextFirstLast

PrevPage: event triggered when user selects to go to the previous page when the controller pager is set to PrevNext or PrevNextFirstLast

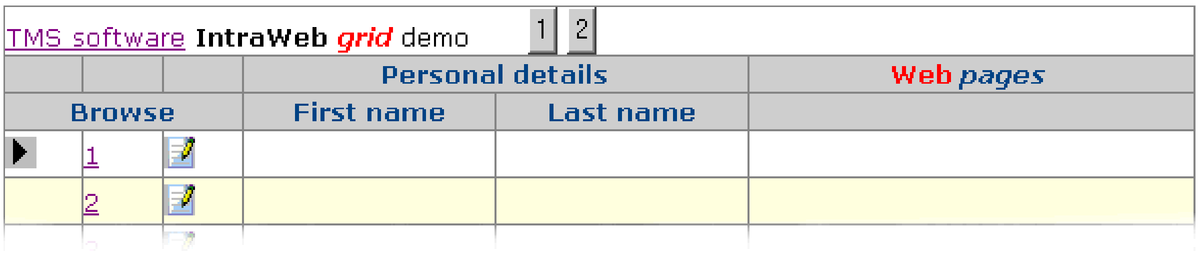
Other related paging properties:

A few other public and published properties are provided here that affect the controller's paging or are affected by it.

StartPage: this provides a programmatic access to set or get the displayed page

Header

Enabling the column headers is done by the property HeaderVisible. All other column header related settings are done through the Columns property. The Columns property is a collection of Column objects that control the appearance of each column in the grid.



Example of what is possible in the header

When the global property **HeaderVisible** is true, following properties in each Column determine what the appearance of the headers will be in the browser :

ColumnHeaderCheck: when true, a checkbox is displayed in the header. Using a checkbox in a column header only makes sense if the column type is CheckBox (see later) . If a checkbox is present in the column header, checking this checkbox will check all checkboxes in the column. Unchecking this checkbox will uncheck all checkboxes in that column.

ColumnHeaderClick: when true, text in the column header is displayed as a hyperlink and clicking this link triggers the event OnColumnHeaderClick. Most commonly, with a databound grid, a SQL statement for sorting the grid can be modified in this event handler. In the non data-aware grid, server side sorting is automatically performed

ColumnHeaderNode : when true, a node in the header will expand or collaps all nodes in the column.

HeaderAlignment: sets the alignment of the column header text
HeaderColor: sets the background color for the column header. Also specifies the start color of a gradient color in the columnheader. When Color Empty is set, no gradient is used. Note that gradients are only supported in IE6.
HeaderColorTo: specifies the end color of a gradient color in the columnheader.
HeaderFont: sets the font for the column header
HeaderGradientOrientation: sets the direction for the gradient to either vertical or horizontal
HeaderVAlignment: sets the vertical alignment for the title

SubTitle: sets the text of the second columnheader row. The second columnheader row is generated as soon as at least on SubTitle is a non empty text.

SubTitleSpan: sets the number of cells this subtitle spans. Note that if SubTitleSpan is set to a value 2 or more, the SubTitle properties in the consecutive columns are ignored.
Title: sets the text of the first columnheader row
TitleRowSpan: Set this to true if the title and subtitle cell must be displayed as merged
TitleSpan: sets the number of cells this title spans. Note that if TitleSpan is set to a value 2 or more, the Title properties in the consecutive columns are ignored.

Example:

For the sample header image above, the following Title, TitleSpan, SubTitle and SubTitleSpan properties were set for each column:

Column 0:
Title = “”, TitleSpan = 0, SubTitle = “Browse”, SubTitleSpan = 3

Column 1:
Title = “”, TitleSpan = 0, SubTitle = “Browse”, SubTitleSpan = 0

Column 2:
Title = “”, TitleSpan = 0, SubTitle = “Browse”, SubTitleSpan = 0

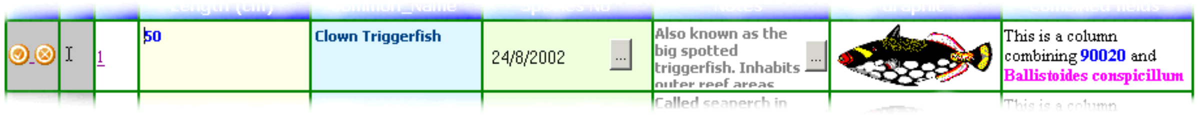
Column 3:
Title = “Personal Details”, TitleSpan = 2, SubTitle = “First name”, SubTitleSpan = 0

Column 4:
Title = “”, TitleSpan = 0, SubTitle = “Last name”, SubTitleSpan = 0

Column 5:
Title = “<FONT color=’#FF0000’>Web </FONT><I>pages</I>”, TitleSpan = 0, SubTitle = “”, SubTitleSpan = 0

Column data display and column types

AdvWebGrid allows other than displaying information from a dataset or cell contents, display of various grid control elements. This can range from simple text cells, hyperlinks to row numbers, DB edit / post / cancel buttons and much more…



I	1	50	Clown Triggerfish	24/8/2002	Also known as the big spotted triggerfish. Inhabits outer reef areas		This is a column combining 90020 and Ballistoides conspicillum

Example of column types

The above example already shows various of these types such as from left to right: DB buttons, DB state indicator, row number link, numeric edit control, text, datepicker, popup memo field, graphic.

Column types:
To explain the various capabilities and how these can be set, it should be noted that a grid cell can be set to 4 different main types :

- *always static cell* : this cell can not be edited and just displays fixed data
- *an action cell* : this cell allow a fixed action, such a handling a button click
- *editable cell* : when the grid is in editing mode, a cell editor is displayed otherwise the cell displays data. Note that setting the grid in editing mode, sets one full row in editing mode at a time. As a result of this, a connection to the server is only done once per row when editing starts and once when editing ends. All cells of a row are thus updated at the end of editing in a single action.
- *dynamic cell* : this type of cells is either always in editing mode or displays a client side calculated value.

The settings for these cell types are done per column through the property Column.Type which currently has following capabilities:

Normal: column contains normal text cells. It is the Editor property that determines whether this column can be edited and if so, what type editor is used.
NoWrap : column contains normal text cells. The column's width does not adapt to the length of the text in the cells. Text that is longer than the column's width is shown with hidden overflow.
Color : column contains a rectangle in the color as set in the cell value, for example AdvWebGrid[0,0] = "#ff0000" will show a red cell.
Checkbox: column contains row select checkboxes
Button: column contains a button. Button caption is set with the ButtonText property
RowIndicator: column contains a glyph that shows for the current row the browse, edit or insert state
RowNumber: column shows the row number
Scroll: column shows text in a scrollbox. This is suitable for memo fields.
Password : columns's text is shown with password characters
Popup: column shows text in cell with button showing text in popup when clicked
PopupImage: the image is displayed in full size (full size is set with PopupWidth and PopupHeight properties) when mouse is over the image, otherwise the image height is limited to the row height.
Progress: the value of the column (between 0 and 100) is displayed as a progress bar
LinkField: column shows text with hyperlink. Clicking the hyperlink moves the current row to the row clicked
LinkRowNumber: column shows the row number as a hyperlink. Clicking the hyperlink moves the current row to the row clicked
DataCheckbox: checkbox state reflects the cell value. Checkbox is checked when cell value is equal to the CheckTrue value or unchecked when equal to CheckFalse
SelectCheckBox : columns shows a checkbox that can be used to select the row (like in Hotmail)
Image: column shows an image
ImageCheckbox: column shows checkboxes with custom glyphs set with the properties ImgCheckBox and ImgCheckBoxChecked.
URL: columns shows text as hyperlink. If text has not yet a http:// prefix, it is automatically inserted
Email: column shows text as email hyperlink. The mailto: prefix is automatically added
DataButton: column shows Edit, Post, Cancel buttons in current row depending on dataset state. Can be set to show Bitmaps or Hyperlinks instead of buttons with the DataButtonType property
RadioButton: columns shows a radio button. Only one radiobutton can be selected per column, allowing row selection through a radiobutton
DynEdit: column shows edit control in all cells. Type of the edit control is set by the property DynEditor
DynText: column shows dynamic text in all cells. Value of the dynamic text is calculated by the Formula property.
DynCheckbox: column shows checkbox in all cells. A checkbox change causes a dynamic value update. The checkbox state can be used in formulas. The value is 1 for a checked checkbox and 0 for unchecked.
DynCombo: column shows combobox in all cells. Combobox items are set with the ComboItems property. A combobox selection change causes a dynamic value update.
DynDatePicker : column shows a datepicker in all cells of the column.
DynDateEdit : columns shows a date-edit entry field in all cells of the column
DynTimeEdit : columns shows a time-edit entry field in all cells of the column
DynLookupEdit : columns shows a editor with lookup. Values used for the lookup are set through the Columns ComboItems collection
DynMemo : columns shows a memo editor in all cells of the column
Node: the column shows a node that is used to hide or unhide a detail row. The settings for the nodes are grouped under the grids Nodes property.

Inplace editors:

The inplace editors are displayed for the current row in the dataset (or ActiveRow for a non data-bound grid). Currently, following inplace editor types are defined:

None: no editor is used in this column, ie. the column is read-only
Edit: inplace editor for column allows any text to be entered
Password: inplace editor is a password style edit
Combo: inplace editor is a combobox. The values for the combobox are set through the stringlist property ComboItems
ColorComboBox : inplace editor is a combobox showing colors
Memo: inplace editor is a textarea
Checkbox: inplace editor is a checkbox
EditNumeric: inplace editor is an edit control that only accepts characters 0..9 and sign.
EditNumericUnsigned: inplace editor is an edit control that only accepts characters 0..9
EditFloat: inplace editor is an edit control that only accepts characters 0..9, a decimal separator and sign.
EditFloatUnsigned: inplace editor is an edit control that only accepts characters 0..9, and a decimal separator. The decimal separator is set with the DecimalSeparator property
EditLower: inplace editor is an edit control automatically converting entered characters to lowercase
EditUpper: inplace editor is an edit control automatically converting entered characters to uppercase
EditHex: inplace editor is an edit control that accepts characters 0..9 and A..F
DatePicker: inplace editor is a datepicker. The format of the datepicker is set with the properties DateFormat and DateSeparator.
DateEdit : inplace editor is a masked date entry editor with three areas for day / month / year
TimeEdit : inplace editor is a masked time entry editor with three areas for hour / min / sec
SpinEdit: inplace editor is a spin edit control
PopupEdit: editing is done through a popup memo editor
LookupEdit : inplace editor performs lookup based on values in the ComboItems collection

Changing inplace editors dynamically:

Normally, cell editor types are set once through the Columns property for each column in the grid. In some cases, it can be convenient to dynamically modify the editor type depending on some cell or row conditions. This can be done with the GetCellEditor delegate. The CellEditorEventArgs object returns the row and column for which the GetCellEditor delegate was triggered and allows to set a new type with the ColumnEditor property.

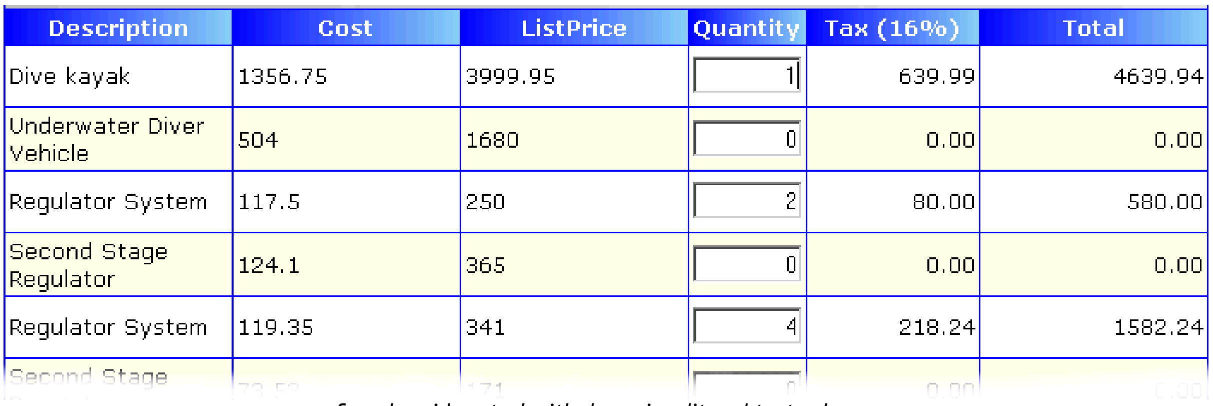
Example:

This code snippet dynamically sets a column editor to a combobox:

```
private void AdvWebGrid2_GetCellEditor(object sender,
TMSWebControls.CellEditorEventArgs e)
{
  e.ColumnEditor = TMSWebControls.Column.ColumnEditor.Combo;
}
```

In addition, it is also possible to change the contents of combobox editors dynamically. This is done through the GetComboItems delegate. The CellComboItemsEventArgs object returns row and column for which the GetComboItems delegate was triggered and allows to change the items of the combobox through ComboItems collection property of this object.

Dynamic edits and text:



Description	Cost	ListPrice	Quantity	Tax (16%)	Total
Dive kayak	1356.75	3999.95	1	639.99	4639.94
Underwater Diver Vehicle	504	1680	0	0.00	0.00
Regulator System	117.5	250	2	80.00	580.00
Second Stage Regulator	124.1	365	0	0.00	0.00
Regulator System	119.35	341	4	218.24	1582.24
Second Stage	73.53	171	0	0.00	0.00

Sample grid control with dynamic edit and text columns

Dynamic edit columns and dynamic text columns allow configuring both the data-bound and not data-bound cells to perform calculations on the client side. In the example above, the Quantity column contains dynamic edit controls, the column Tax and Total contain dynamic text. The values in the dynamic text columns are dependent on the values in the ListPrice column and the Quantity column.

The formula for calculating the tax is:

Tax = 0.16 * Quantity * ListPrice

The formula for calculating the total is:

Total = 1.16 * Quantity * ListPrice

Configuring this in AdvWebGrid is as simple as setting the Quantity column type with ColumnType to DynEdit and setting the Tax and Total columns ColumnType to DynText. Next, the formula needs to be set that calculates the dynamic text columns. This is done with the Formula property for each column. A formula expression can be written using the variables C1, C2, ..., Cn, where Cx is the variable holding the value of the cell in column x.

In this example for tax and total calculation, the formulas for column Tax and Total are :

Tax column:

Formula = C3*C2/100*16

Total column:

Formula = C2*C3*1.16

Two more properties are used to control dynamic editing and dynamic text. First property is the DynEditor property. This property can be set to:

Text: allow any text input in a dynamic
Unsigned: allow unsigned numeric input
Signed: allow signed numeric input
FloatUnsigned: allow unsigned floating point input
FloatSigned: allow signed floating point input

Finally, the DynPrecision property controls the number of decimals to display in the calculated result of a dynamic text.

Retrieving or presetting the values of dynamic edit controls is simple. It can be accessed with the grid[AColumn,ARow] property. Thus, presetting a dynamic edit for column 3, row 7 can be done by:

Grid[2,6] = “1234”; (note that column and row indexes are always zero based)

After a submit, the server side can retrieve the edited value with the same Grid[2,6] property.

Note that presetting or retrieving values of dynamic text is not available. As dynamic text is always calculated with a known formula, the server side can at any time, based on database field values and dynamic edit values know the value of dynamic text.

It is possible to programmatically change or generate column data on the fly. This is done through the event GetCellData. This event is defined as:

```
private void AdvWebGrid1_GetCellData(object sender,
TMSWebControls.GetCellDataEventArgs e)
```

With the following attributes:

```
Int e.Row
Int e.Column
String e.Value
```

The event is triggered for each cell rendered for the browser. It allows changing the data dynamically on the server before being sent to the browser. Shown here is a sample that simulates Windows style ellipsis drawing for large text for column 4:

```
private void AdvWebGrid1_OnGetCellData(object sender,
TMSWebControls.GetCellDataEventArgs e)
{
if (e.Column = 4)
     {
           if (Length(e.Value) > 15)
               e.Value = e.Value.SubString(1,15) + "…";
     }
}
```

Column widths:

Although different width specifications for columns are possible : none, percent, absolute, the absolute width specification is recommended as it allows exactly positioning the grid control with other controls on the form. When setting the column's width type to Absolute, the width is set as pixel width with the Width property.

Programmatically changing column settings:

All column information is stored in an array AdvWebGrid.Columns. Accessing an element of this array is done with AdvWebGrid.Columns[index]. In order to access the properties, cast this to a TMSWebControls.Column object.

To change the width, the code that can be used is:

```
((TMSWebControls.Column)AdvWebGrid.Columns[ColumnIndex]).Width = 40;
```

Footer

The grid's footer is much like the grid's header. It is displayed when the property ShowFooter is true. The properties that determine the appearance of the footer are set per column in the Columns property:

FooterColor: sets the background color for the column header. Also specifies the start color of a gradient color in the columnheader. When Color Empty is set, no gradient is used. Note that gradients are only supported in IE6.
FooterColorTo: specifies the end color of a gradient color in the columnheader.
FooterFont: sets the font for the column header
FooterGradientOrientation: sets the direction for the gradient to either vertical or horizontal
FooterVAlignment: sets the vertical alignment for the title
FooterAlignment: sets the alignment for the footer text
FooterFormat: is a format specifier for calculated footer values
FooterText: holds the fixed text for a footer
FooterType: the footer for each column can have following types :

Text: footer contains simple static text
PageSum: footer contains server calculated sum of column cell values of page
PageMin: footer contains server calculated minimum of column cell values
PageMax: footer contains server calculated maximum of column cell values
PageAvg: footer contains server calculated average of column cell values
None: footer is empty
DynSum: footer contains client side calculated sum of column cell values
DynMin: footer contains client side calculated minimum of column cell values
DynMax: footer contains client side calculated maximum of column cell values
DynAvg: footer contains client side calculated average of column cell values

Note that dynamically generated values only make sense for columns that have the DynText or DynEdit style. For other column types, the column cell values are not dynamically updated in the browser, thus recalculating columns should never be done in the browser.

Note also that the format of the output for server side calculated footers can be set trough the FooterFormat property.

Example:

FooterFormat = “Average : 0.00”;

This shows the average of values displayed with PageAvg type with 2 decimals.

For dynamically calculated footers, the DynPrecision property determines the number of decimals that will be displayed.

RowHeader

When the RowHeader is enabled, a new column is added before the first column which displays the rownumber. This is a fixed column so the rownumbers remain visible when using vertical scrolling. The RowHeader’s appearance can be controlled with the RowHeader subproperties:

BackColor: sets the background color for the row header. Also specifies the start color of a gradient color in the rowheader. When Color Empty is set, no gradient is used. Note that gradients are only supported in IE6.
BackColorTo: specifies the end color of a gradient color in the columnheader.
Font: sets the font for the column header
GradientOrientation: sets the direction for the gradient to either vertical or horizontal
Visible: specifies if a rowheader is displayed or not
Width: sets the width of the rowheader

Borders

Full Border configuration is available for all the AdvWebGrid components. Configuring borders is similar to setting borders for a Table in HTML. The properties are separate so they also have to be set seperately for the ColumnHeader, the Footer, the Controller, the Grid Cells and the RowHeader.

Collapse:
Color: set the color of the borders
ColorDark: set the color of the borders
ColorLight: set the color of the borders
Inner: specify which inner borders should be displayed (all, rows, columns, none)
Outer: specify which outer borders should be displayed
void: none
border: up-down-left-right
below: down
hsides: up-down
lhs: left
rhs: right
vsides: left-right
box: up-down-left-right
Padding: specify the width between the cell border and the cell content
Spacing: specify the width between two cells
Width: sets the width of the borders

Cell and row selection

Checkbox disjunct multi-row selection:

The most intuitive and familiar interface for selecting rows in a grid is perhaps the interface we have all learned from Hotmail, the checkbox based selection. This is enabled in AdvWebGrid by adding a column with the CheckBox style (optionally with ColumnHeaderCheckbox true, a checkbox in the column header will select / unselect all checkboxes) If a checkbox is checked, the row is selected and displayed in the SelectColor / SelectFontColor. The selection is fully handled on the client side. Only upon a server connection, the server application can get the state of the selected rows with the property bool Selected[int RowIndex].

Following code shows how selected rows in a single displayed page can be deleted:

```
AdvWebGrid1.RemoveSelectedRows();
```

Optionally, an event is triggered OnCheckClick. Note that when the OnCheckClick event is assigned, a connection to the server will be made for each checkbox click.

Selections by mouseclick on cells:

It is not required to use checkboxes for selecting rows. Using the property MouseSelect, following selection methods can be used:

Row : single click selects row, another single click unselects current row and selects new row.
SingleCell: single click selects single cell and unselects previously selected cell. Optionally, an event is triggered OnCellClick. Note that if this event handler is assigned, for each cell click a server connection will be made.
RowCheck: single click selects row as if click on a checkbox. With SelectPersistent, the selected row state is remembered across pages and can be retrieved or set with Bool grid.Select[int RowIndex];
Move: single click moves the database cursor to the clicked row
Client: generates a client side event only. The event can be handled with code in the ClientEvents.CellClick property
None: no events are triggered for clicks on normal grid cells

Finally, if the property AutoEdit is set true, the grid will automatically switch to editing mode, when a mouse click happens on a selected row.

Row coloring

By default, cell colors are set through the Color property for each Column. In addition, the GetCellProp event allows dynamic and/or content based cell and/or row color changing.Often it is much more convenient, to quickly apply a few often used coloring schemes. In AdvWebGrid, these are color banding, selection colors, edit color and hovering:

Color banding:
This is the often used alternating color per row scheme. It is simply enabled by setting the property grid.Bands.Active to true and defining colors for odd and even rows through grid.Bands.PrimaryColor and grid.Bands.SecondaryColor.

Selection colors:
The color of selected rows is set by SelectColor and SelectFontColor properties.

Active row color:
The color of the active row (current DB record for the DB-aware version or ActiveRow for non DB-aware version) can be set with ActiveRowColor and ActiveRowFontColor properties.
When the grid is in editing state, the row that is being editing can be conveniently displayed in its own color set by ActiveRowColor.

Hovering:
Hovering is the effect that a row changes color when the mouse is over it. It is enabled in AdvWebGrid setting the properties HoverColor and HoverFontColor.

General note:

When setting these color properties to Color Empty, the selected row coloring schemes are not used.

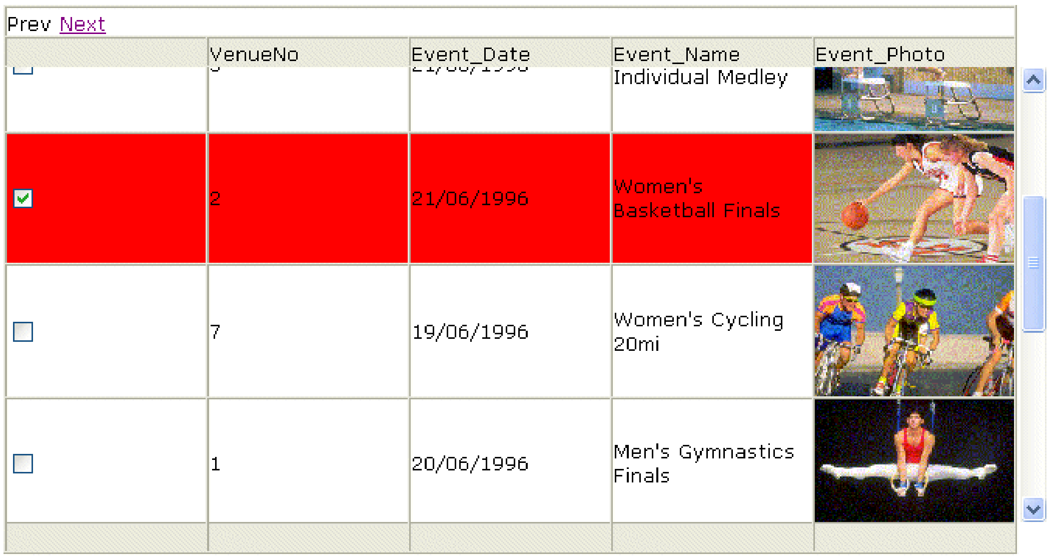
Sort control

A data-bound AdvWebGrid can use the built-in sorting or rely on the DB capabilities to sort the dataset shown in the grid. A non data-bound AdvWebGrid has built-in sorting which is enabled by global setting grid.SortSettings.Column to the column that should be sorted and further enabled for only those columns that must be allowed to be sorted by setting ColumnHeaderClick to true. For AdvWebGrid, the normal procedure to handle sorting is writing an event for the ColumnHeaderClick, change the query statement in this event handler to sort for the clicked column or when the clicked column was previously sorted, toggle the sort direction. The grid will then visually indicate the sort direction of the sorted column by a small up or down arrow in the column header. The same applies for AdvWebGrid. The difference here is that AdvWebGrid performs sorting internally and takes care of the SortSettings property by updating sorted column and sort direction for each click on a column header. As the AdvWebGrid performs its own sorting, the type of data displayed in each column must be set to allow the internal sort to work with the correct compare routines. This is set with the property SortFormat available in AdvWebGrid only and can be:

Alphabetic : alphabetic sorting
Numeric : integer or floating point based sorting
Date : date based sorting

Built-in scroll support

AdvWebGrid has built-in scroll support. This means that you can add scrolling capabilities to the grid by just setting one property : grid.Scroll.Enable = true



Prev Next

	VenueNo	Event_Date	Event_Name	Event_Photo
☐			Individual Medley	
✓	2	21/06/1996	Women's Basketball Finals	
☐	7	19/06/1996	Women's Cycling 20mi	
☐	1	20/06/1996	Men's Gymnastics Finals	

The scrolling is enabled for the data rows of the grid. An additional property is available: grid.Scroll.Persistent. When this is true, the vertical scroll position is persistent between consecutive rendered pages, ie. the grid remembers and restores the last scroll position for each new renderering after a server connection.

It is also possible to control programmatically the scroll position of the grid. This can be convenient to force the grid to scroll to a given position to bring into view active rows or rows that have been changed. This can be done using the property grid.Scroll.Position. This property is a value between 0% and 100%. When scroll persistence is disabled and scrolling is enabled, setting grid.Scroll.Position to 50 will show the grid half-way scrolled.

Additionally, under grid.Scroll various scroll bar color settings are available.

Using detailrows

Detailrows offer the capability to show additional record information only when the user selects to open this.

Detailrows have common properties that are:

DetailRowHeight: sets the height of the detailrow. If this is 0, the height automatically adapts to the information in the detail row.

DetailRowShow: selects the method to display the detail row. This can be:

Normal: client side opening of a detail row with a node without affecting other detail rows
OneOpen: client side opening of a detail row with a node with automatic closing of other detail rows to make sure only one detail row is open at a time
AllOpen: grid is rendered with all detail rows immediately open
ServerOpen: detail row is opened after a server connection. During the server connection, the DetailRowOpen, DetailRowClose events are generated when a new detail row has been opened.

ServerOneOpen: detail row is opened after a server connection. During the server connection, the DetailRowOpen, DetailRowClose events are generated when a new detail row has been opened. Previously opened detail rows are automatically closed upon opening a new detail row.

Other settings for detailrows are in the Columns property:

DetailColor: sets the background color of the detail row in this Column

DetailForeColor: sets the text color of the detail row in this Column

DetailField : sets optionally the DB field that should be shown in the detail row

DetailFont : sets the font to be used in the detail row in this Column

DetailSpan: sets the number of columns the detail row spans from this column

Additionally, for each row the event GetDetail is also triggered to set the detail row information dynamically.

Detail rows are opened or closed with nodes. It is therefore required that at least one column in the grid contains nodes. A column can be set to contain nodes when its columntype is Node. The settings for the node appearance are available in the Images property with:

NodeCloseHint: set the hint to display when the mouse is over a closed node
NodeOpenHint: sets the hint to display when the mouse is over an opened node
NodeClose: sets the glyph to display for a closed node
NodeOpen: sets the glyph to display for an open node

The detailrows feature persistence, preset and checking state. Persistence means that a client side open or close of a detail row is persistent across consecutive rendered pages after a server connection. The state of a detail row can be set and checked with the public property:

grid.DetailStates[Int Row] = true;
grid.DetailStates[Int Row] = false;

Example:

This code sets up an AdvWebGrid with a detailrow that spans a full row and that presets the first five rows to have the detailrow open:

```
for (int i=0;i<5;i++)
    AdvWebGrid1.DetailStates[i] = true;
```

Using the server side opened and closed detailrows can be used to optimize bandwidth. By using this DetailRow mode, only the data for opened detailrows will be sent to the browser and thus avoiding the sending of a lot of potentially never seen data to the browser.

Using internal cell methods in AdvWebGrid

The data in AdvWebGrid is set through the this[col,row] property. Various methods exist to handle the cells:

property string this[ACol,ARow: Integer];
Basic interface through which cell data can be set.

procedure ClearCells;
Clears contents of all cells

procedure DeleteRows(RowIndex,RowCount: Integer);
Deletes RowCount rows starting from RowIndex

procedure InsertRows(RowIndex,RowCount: Integer);
Inserts RowCount rows at position RowIndex

procedure InsertColumns(ColIndex, ColCount: Integer);
Inserts ColCount columns at position ColIndex

procedure DeleteColumns(ColIndex, ColCount: Integer);
Deletes ColCount columns starting from column ColIndex

procedure ClearRows(RowIndex,RowCount: Integer);
Clears contents of RowCount rows starting at row RowIndex

procedure ClearColumns(ColIndex,ColCount: Integer);
Clears contents of ColCount columns starting at row ColIndex

Advanced AdvWebGrid topics

Using the clientevents

It is possible to write JavaScript code that is handled in the browser for following events:

Button click : ButtonClick
Cell Click : CellClick
Combobox Change : ComboChange
Dynamic checkbox click : DynCheckClick
Dynamic combobox change : DynComboChange
Dynamic edit change : DynEditChange
Dynamic end of edit : DynEditDone
End of edit : EditDone
Image click : ImageClick
Node click : NodeClick

The JavaScript code is added as a StringList in the ClientEvents property. This code is inserted inside the event handlers as JavaScript and will be executed in the browser. Note that any error in the JavaScript code can potentially cause that the grid is no longer working correct and that JavaScript errors are displayed in the browser.

In JavaScript, a number of functions are available that can be used. To call these methods, it is important to prefix ALL the functions with “GridName”Obj where GridName is the grid’s name.

Available methods:

GridNameObj.GetEditRow(): returns the row index of the currently edited row mode.

GridNameObj.IsEditing(); returns true if the grid is in editing mode

GridNameObj.GetCellValue(c,r); returns the value of cell c,r. This is only applicable for cells that have text and not with cells that have controls or images.

GridNameObj.SetCellValue(c,r,value); sets the value of cell c,r.

GridNameObj.GetEditValue(c,r); returns the value of the edit control in cell c,r

GridNameObj.SetEditValue(c,r,value); set the value of the edit control in cell c,r

When the clientside events are called, the variables c & r indicate the cell for which the event was triggered. Depending on the event, additional parameters are available.

Example: presetting values for grid in edit mode

The code below is added to the event for a button click. It first checks if the grid is in editing mode. If so, it sets the values of the inplace editors to preset values:

```
if (!AdvWebGrid1Obj.IsEditing())
{
alert("Cannot preset values : not in editing mode");
return;
}
i = AdvWebGrid1Obj.GetEditRow();
AdvWebGrid1Obj.SetEditValue(3,i,"Danny");
AdvWebGrid1Obj.SetEditValue(4,i,"Thorpe");
AdvWebGrid1Obj.SetEditValue(5,i,"Borland");
```

Note the calls to AdvWebGrid1Obj.IsEditing(), where AdvWebGrid1Obj is the reference to the grid in the browser)