Exploring the Async capabilities of TMS IntraWeb grids

For as long as web development is being done, developers have tried to make the experience of using web applications as similar as possible as using native Windows applications. Initially, this was done by delegating all behavior that didn't require direct interaction with the server to Javascript code running in the browser. In 1999, Microsoft added the XMLHTTP ActiveX control as standard component in Internet Explorer 5. It took some time till the XMLHttpRequest object was being heavily used in all kinds of web applications. With XMLHttpRequest, it was possible to send data to the server and receive data back without requiring a full page refresh. The most well-known early large scale adaptors of this technology were Outlook Web Access in 2000, Google's GMail in 2004 and Google Maps in 2005. Around this time, the techniques to asynchronously send and receive data from the server were categorized under the umbrella AJAX.

For Delphi developers, the IntraWeb framework from Atozed nicely encapsulates all technical wizardry that is needed to make XMLHttpRequest work to asynchronously send data to the server and asynchronously update parts of a web page. As RAD as we can expect from a Delphi framework, other than handling Async events in Delphi code, there is not much extra to do.

Introduction to the very basics of Ajax in IntraWeb

To show how really simple using the asynchronous capabilities of IntraWeb are, drop a TIWLabel, TIWEdit and TIWButton on the form and handle the TIWButton's OnAsyncClick event. In this event handler, add the code:

```
procedure TIWForm4.IWButton1AsyncClick(Sender: TObject;
  EventParams: TStringList);
begin
  IWLabel1.Caption := IWEdit1.Text;
end;
```

When the button is clicked, the content of the edit control is asynchronously sent to the server and the text of the label is asynchronously updated. IntraWeb has taken care to send the IWEdit1 text asynchronously to the server, trigger the OnAsyncClick event handler code server side and send back the instruction to update just the label's caption. Given that a minimal amount of data is used to accomplish this, the update of the label is blinking fast and behaves almost identically to a native Windows application. In a similar way, most of the basic IntraWeb components offer one or multiple asynchronous events and support updating components asynchronously.

Introduction to TMS IntraWeb grids

TMS software offers two grid components for IntraWeb. A non DB-aware grid TTIWAdvWebGrid and a DB-aware grid TTIWDBAdvWebGrid. As TTIWDBAdvWebGrid descends from the same internal TTIWCustomWebGrid as TTIWAdbWebGrid descends, the capabilities and features of both components are very similar. The capabilities far surpass these of the standard IntraWeb grid component. There are a lot more ways to show data in the grid and a lot more built-in behaviors like for example sorting and paging. The grids have built-in capabilities to update itself fully client-side or with fast asynchronous server interaction. The choice is left to the developer what specific features will be used. Of course it makes sense that when a specific behavior can be obtained without any server interaction at all, this is even faster. Although the article focuses on using the asynchronous capabilities of the grid, a short example of using the fully client-side built-in behaviors in the grid is given to show the difference. To see this behavior, drop a default TTIWAdvWebGrid on the form and add the following code:

```
procedure TIWForm4.IWAppFormCreate(Sender: TObject);
begin
  TIWAdvWebGrid1.ShowFooter := true;
  TIWAdvWebGrid1.Columns[0].Title := 'Quantity';
  TIWAdvWebGrid1.Columns[0].FooterType := ftDynSum;
  TIWAdvWebGrid1.Columns[0].ColumnType := ctDynEdit;
  TIWAdvWebGrid1.Columns[1].ColumnType := ctDynText;
  TIWAdvWebGrid1.Columns[1].Formula := 'C0*1.19';
  TIWAdvWebGrid1.Columns[1].DynPrecision := 2;
  TIWAdvWebGrid1.Columns[1].FooterType := ftDynSum;
  TIWAdvWebGrid1.SortSettings.Show := true;
end;
```

The result is that in all cells in the first column, an edit control is shown and that in the footer, the sum of the values in the column is shown. In the 2nd column, a number is shown that is the result of the calculation of the value in column 0 multiplied by 1.19, for example to calculate a price including TAX and in the footer the sum of the prices with TAX included. As the user types numbers in column 0, the value in the footers and second column are updated without any server interaction.

Asynchronously updating the grid

The grid has two methods to update cell values asynchronously:

```
grid.AsyncSetCell(Col,Row,Value);

grid.AsyncUpdateAllCells;
```

The first method updates just one cell, the second method updates the content of all cells as set with grid.Cells[Col,Row]:string asynchronously.

To demonstrate this, add a default TTIWAdvWebGrid, TIWEdit, TIWLabel and TIWButton on the form with following code:

```
procedure TIWForm4.IWAppFormCreate(Sender: TObject);
begin
  TIWAdvWebGrid1.ActiveRow := 0;
  TIWAdvWebGrid1.ActiveRowColor := clInfoBk;
  TIWAdvWebGrid1.AsyncActiveRowMove := true;
end;

procedure TIWForm4.IWButton1AsyncClick(Sender: TObject;
  EventParams: TStringList);
var
  ar: integer;
begin
  ar := TIWAdvWebGrid1.ActiveRow;
  IWLabel1.Caption := 'Value for row: ' + inttostr(ar);
  TIWAdvWebGrid1.AsyncSetCell(0,TIWAdvWebGrid1.ActiveRow,IWEdit1.Text);
  TIWAdvWebGrid1.ActiveRow := ar + 1;
end;
```

In this sample, the active row of the grid is indicated in color as set with grid.ActiveRowColor and setting grid.AsyncActiveRowMove = true means that changing the property grid.ActiveRow will asynchronously update the highlighted active row in the browser. In the asynchronous event of the button, the value of the cell in column 0 in the active row is set and the active row is moved one row further. In the case the grid is populated with a lot of data, possibly images, controls etc... it is obvious that asynchronously just updating a cell value is significantly faster than having to rerender the entire grid during a page refresh.

Asynchronously sorting the grid

Typically in grids, sorting on a specific column is done by clicking the column header. When grid.SortSettings.Show is set to true, sorting by such column header click is enabled. A small triangle is shown on the column that is sorted indicating the sort direction. To enable sorting asynchronously, set grid.AsyncSorting = true and specify the way the column should be sorted with grid.Columns[index].SortFormat. Clicking on the column title will then toggle the sort direction on each click and update the entire grid with sorted data asynchronously.

Example:

```
procedure TIWForm4.IWAppFormCreate(Sender: TObject);
const
  strarr : array[0..9] of string =
('BMW','Audi','Porsche','Mercedes','VW','Ferarri','Lamborghini','Porsche','Aston
Martin', 'Maseratti');
var
  i: integer;
begin
  for I := 0 to 9 do
    TIWAdvWebGrid1.Cells[0,i] := strarr[i];
  TIWAdvWebGrid1.AsyncSorting := true;
  TIWAdvWebGrid1.SortSettings.Show := true;
  TIWAdvWebGrid1.Columns[0].Title := 'Brand';
  TIWAdvWebGrid1.Columns[0].SortFormat := sfAlphabetic;
end;
```

Asynchronous paging in the grid

Paging is a common technique when showing large amounts of data in tabular form in web applications. To reduce the amount of data that needs to be sent to the browser, data is sent page by page. The TMS grids have built-in paging capabilities. How paging is done is configured in the footer. Different types of paging can be chosen, like showing a list of all pages in the header, show just a Next/Previous button and many more options. The number of rows per page is set with grid.RowCount and the total nr. of rows is set with grid.TotalRows.

There is just one property, grid.AsyncPaging to set to true, and walking through the different pages of the grid is done fully asynchronously. When clicking in the header to navigate to a different page, only the cell values for the new page are being sent from the server. The sample code snippet shows how a default TTIWAdvWebGrid is setup to perform asynchronous paging:

```
procedure TIWForm4.IWAppFormCreate(Sender: TObject);
var
  i: integer;
begin
  TIWAdvWebGrid1.Controller.PagerType := cptLink;
  TIWAdvWebGrid1.Controller.Pager := cpPageList;
  TIWAdvWebGrid1.Controller.MaxPages := 4;
  TIWAdvWebGrid1.Controller.IndicatorFormat := 'Page %d';
  TIWAdvWebGrid1.RowCount := 10;
  TIWAdvWebGrid1.TotalRows := 100;

  for I := 0 to 99 do
    TIWAdvWebGrid1.Cells[0,i] := 'Row: ' + inttostr(I);

  TIWAdvWebGrid1.AsyncPaging := true;
end;
```

Asynchronous editing in the grid

Just like when using a native DB grid, typically a record (row) is put in edit mode and when row editing is done, the edited values are posted back to the database. The TMS IntraWeb grids have this behavior built-in for both the DB-aware and non DB-aware grid. Either the grid can be programmatically set in edit mode or programmatically post data, or one column can be configured as a data button column. This means that when grid.Columns[index].ColumnType is set to ctDataButotn, a button appears in the column to set the record (row) in edit mode and when in edit mode, two buttons appear in this column to either post or cancel. When the record (row) is put in edit mode, the inplace editors for this row will appear. For each column a different inplace editor can be chosen. The grid comes default with different edit controls, combobox, datepicker, checkbox, spin edit, memo... This sample shows how easy it is to have the entire process of sophisticated editing working asynchronously in the grid. A default grid is initialized with the code:

```
procedure TIWForm4.IWAppFormCreate(Sender: TObject);
begin
  TIWAdvWebGrid1.Columns.Add;
  // enable async editing
  TIWAdvWebGrid1.AsyncEdit := true;
  // column 0 works as data button column to start editing, post or cancel
  TIWAdvWebGrid1.Columns[0].ColumnType := ctDataButton;
  // set editor types for column 1 & 2
  TIWAdvWebGrid1.Columns[1].Editor := edEdit;
  TIWAdvWebGrid1.Columns[2].Editor := edCombo;
  // set the values for the combobox in column 2:
  TIWAdvWebGrid1.Columns[2].ComboItems.Add('BMW');
  TIWAdvWebGrid1.Columns[2].ComboItems.Add('Audi');
  TIWAdvWebGrid1.Columns[2].ComboItems.Add('Mercedes');
  TIWAdvWebGrid1.Columns[2].ComboItems.Add('Porsche');
end;
```

With this straightforward initialization, a button appears in column 0 that can put a specific record in edit mode. As async editing is enabled, without page refresh, the chosen inplace editors appear as set for each column. The different columns can be edited and when done, either the Post or Cancel button can be clicked to post back the data to the server (and possibly database too). This data is also sent back asynchronously to the server and without any page refresh, the grid's inplace editors disappear and the row returns back to regular browse mode. To go one step further to show how powerful the async capabilities of the grid are, the event OnAsyncComboChange is implemented. This event is triggered when a value is selected in the combobox. From this event, we can for example asynchronously update a grid cell like in the sample code. Here the value of the cell in column 3 is set to the index of the value chosen in the combobox:

```
procedure TIWForm4.TIWAdvWebGrid1AsyncComboChange(Sender: TObject;
  EventParams: TStringList; RowIndex, ColumnIndex: Integer; AValue: string);
begin
  TIWAdvWebGrid1.AsyncSetCell(3,RowIndex,
IntToStr(TIWAdvWebGrid1.Columns[2].ComboItems.IndexOf(AValue)));
end;
```

Async row selection and async row moving

Disjunct row selection, like in Hotmail, is easily enabled in the grid by adding a column with a checkbox. The selection of a row can then be asynchronously handled via the event OnAsyncRowSelect. In this event, other controls or the grid itself can be updated. The sample code snippet, again applied to a default TTIWAdvWebGrid shows how a label is updated with the index of the last selected row and the text of the selected row itself is asynchronously updated to bold font:

```
procedure TIWForm4.IWAppFormCreate(Sender: TObject);
var
  i: integer;
begin
  for I := 0 to 9 do
    TIWAdvWebGrid1.Cells[1,i] := 'Row ' + inttostr(i);
  TIWAdvWebGrid1.Columns[0].ColumnType := ctCheckbox;
end;

procedure TIWForm4.TIWAdvWebGrid1AsyncRowSelect(Sender: TObject;
  EventParams: TStringList; RowIndex: Integer; Checked: Boolean);
begin
  IWLabel1.Caption := 'Selected row: ' + IntToStr(RowIndex);
  if Checked then
    TIWAdvWebGrid1.AsyncSetCell(1,RowIndex,'<B>'+
TIWAdvWebGrid1.Cells[1,RowIndex]+'</B>')
  else
    TIWAdvWebGrid1.AsyncSetCell(1, RowIndex,
      Copy(TIWAdvWebGrid1.Cells[1,RowIndex],4,
Length(TIWAdvWebGrid1.Cells[1,RowIndex]) - 7));
end;
```

In the cases where only one row can be selected at a time, for example by just clicking the mouse on a row to set the active row, this can also be entirely asynchronously handled:

```
procedure TIWForm4.IWAppFormCreate(Sender: TObject);
var
  i: integer;
begin
  for I := 0 to 9 do
    TIWAdvWebGrid1.Cells[0,i] := 'Row ' + inttostr(i);
  TIWAdvWebGrid1.ActiveRowColor := clInfoBk;
  TIWAdvWebGrid1.AsyncActiveRowMove := true;
  TIWAdvWebGrid1.MouseSelect := msMove;
end;
```

Here, with setting grid.MouseSelect = msMove, a click on a row will move the active row. The active row is shown with color clInfoBk. By setting grid.AsyncActiveRowMove = true, the active row will be updated without full page refresh. The change of the active row can be asynchronously tracked server side by handling the event OnAsyncGotoRow. In this event, information in other controls can be updated to reflect the active selected row, ie:

```
procedure TIWForm4.TIWAdvWebGrid1AsyncGotoRow(Sender: TObject;
  EventParams: TStringList; Row: Integer);
begin
  IWLabel1.Caption := 'Active row: '+inttostr(Row)+' Value: ' +
TIWAdvWebGrid1.Cells[1,Row];
end;
```

Conclusion

The TMS IntraWeb grids are packed with many features to operate in asynchronous scenarios. From other IntraWeb controls the grid cells can be updated asynchronously and simultaneously, many of the standard grid operations like editing, paging, sorting, navigation can be fully handled asynchronously as well. By combining these capabilities, very high performance web applications can be created that offer a near native application experience.