Time Related Performance Issues

Remember to update to the newest version of OfficeWriter! New updates are periodically released which include new features, tweaks to improve usability, and bug fixes to help make your OfficeWriter experience as productive as possible.

Table of Contents

Use InsertRows/Columns Instead of InsertRow/Column

What might cause this to happen:

- Someone is adding multiple rows into a template file yet is unaware that both InsertRow and InsertRows are legitimate functions (same with InsertColumn(s)).

```
/* INCORRECT CODE:

for (int i = 1; i < 50; i++)
{
     ws.InsertRow(i + 5);
     ws.InsertColumn(i + 5);
} */
```

Why this method causes issues:

- Every time a call is sent for InsertRow, InsertRows, InsertColumn, DeleteRow, etc., it requires the updating of the entire grid to calculate the new position of all rows/columns beyond the adjusted ones.
- Grid updates also require that any formulas that point to something beyond the adjusted position have to be updated to compensate.

Solutions:

- If adding more than just one or two rows, use InsertRows/InsertColumns instead of looping through InsertRow/Column multiple times.
- InsertRow/Column can still be used, but as grid updates are expensive, they should be retained only for calls of one or two new lines.

```
// POSSIBLE CORRECT CODE:

// Example inserting a single row and column to split up data.
ws.InsertRow(1);
ws.InsertColumn(1);

// Example inserting multiple rows and columns to move data down.
ws.InsertRows(0, 10);
ws.InsertColumns(0, 10);
```

Avoid Calling AutoFitWidth on a Large Amount of Data

What might cause this to happen:

- As AutoFitWidth sets the width of a column to just beyond the length of the longest cell, and as such can be seen as a very simple way to format items.

```
/* INCORRECT CODE:

ColumnProperties columnProperties;

for (int i = 0; i < 3; i++)
{
     columnProperties = ws.GetColumnProperties(i);
     columnProperties.AutoFitWidth();
} */
```

Why this method causes issues:

- Since AutoFitWidth has to loop through all of the values in a column to determine which has the longest length, it often becomes very time consuming to call it on larger data sets.

Solutions:

- AutoFitWidth can still be called on very small amounts of data, but if it's even debatable whether a data might be too large, it should be avoided.
- Determine a likely possible max length for populated cells, and adjust column widths beforehand (may not look as nice, but runs much better).
- You can loop through the DataTable manually. Although you are still looping through all the data, DataTables are simpler to loop through than entire worksheets.
- Be wary of the fact that datamarkers allow the order of columns in the data source and in the resulting worksheet to be different. This method only workers if you are sure of a column's location in both the original DataTable and in the output file.
- If estimating, remember to account for any formatting that may be applied, such as currency formatting.

```
// POSSIBLE CORRECT CODE:

DataTable dts = GetData(Page.MapPath(@"data\PersonsInfoV2.csv"));

// Column on which the custom autofit starts, to be set by user.
int startingColumn = int.Parse(dts.Rows[0][9].ToString());

// Column on which the custom autofit ends, to be set by user.  (Should be set to
the row AFTER the last column to be autofit.)
int endColumn = int.Parse(dts.Rows[1][9].ToString());

// Array of the longest variables in each column.
int[] longest;

// Temporary variable for keeping track of current location
int temp = 0;

// The loop to check on the width of each column.  Only start if the given
starting and ending places are valid.
if ((startingColumn < dts.Columns.Count)&&(endColumn <
dts.Columns.Count)&&(startingColumn < endColumn))
{
    // Find the size of the array, based on the starting location.
    longest =  new int[endColumn - startingColumn];

    // For each row in the DataTable...
    foreach (DataRow row in dts.Rows)
    {
        // For each column, based on that row...
        for (int i = startingColumn; i < endColumn; i++)
        {
            // Find the temporary length of that row.
            temp = row[i].ToString().Length;

            // Is it the longest in the column so far?  If yes, set it as such.
(Longest subtracts startColumn from index to keep starting index at 0.)
            if (temp > longest[i-startingColumn])
            {
                longest[i-startingColumn] = temp;
            }
        }
    }

    // After looping through, set the width of each column to the longest.
    // You can change this function to change where in the output file
    for (int i = startingColumn; i < endColumn; i++)
    {
        ColumnProperties columnProperties;
        columnProperties = ws.GetColumnProperties(i);
        columnProperties.WidthInChars = longest[i-startingColumn];
    }
}
```

Cache Frequently

What might cause this to happen:

- Most documents are set up this way by default, as is in many cases it is not much of an issue.

Why this method causes issues:

- A document that is called a lot and is set to run whenever someone calls it may result in issues if it gets to be large enough in size, since it has to retrieve the information each time.

Solutions:

- Check to see if there is new info or not; if there is, update; if not, use last version.
- Save a copy of the Output each time it receives new data to be used until more new data is added.

Memory Related Performance Issues

View the information on performance issues that effect the amount of memory taken up by a report. Even if memory performance issues were your main concern, you may wish to view these fixes as well, as there tends to be some overlap between the categories.