Memory 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

Avoid Opening Workbooks Populated with Large Quantities of Data with ExcelApplication

What might cause this to happen:

- Result of an attempt to get dynamic formatting based on user choices in asp.net.
- Attempts to autofit column widths after the data has been populated.
- To add special formatting to cells based on their values.

```
/* INCORRECT CODE:


ExcelTemplate xlt = new ExcelTemplate();
xlt.Open(Page.MapPath(@"path\to\code.xlsx"));

/*...*/

xlt.Process();

ExcelApplication xla = new ExcelTemplate();
Workbook wb = xla.Open(xlt);

xla.Save(wb, "Completed.xlsx"); */
```

Why this method causes issues:

- ExcelApplication fills out everything needed for customization, so when already populated files are passed to it, it requires more for it to load them in than it does for ExcelTemplate alone.
- Also results in a large hit on Time based performance.

Solutions:

- If the workbook formatting needs to be modified dynamically at runtime, the changes should be made in the unpopulated template before populating the data, rather than to the populated file.
- Design the template file, load it in ExcelApplication and make any other customizations desired, then pass it on to ExcelTemplate to load in the data.

```
// POSSIBLE CORRECT CODE:


ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Open(Page.MapPath(@"path\to\code.xlsx"));

/*...*/

ExcelTemplate xlt = new ExcelTemplate();

xlt.Open(xla, wb);

xlt.Process();
xlt.Save(Page.Response, "Completed.xlsx", false);
```

Issues with Empty Cells

1.) Referencing cells that don't exist in the input file

What might cause this to happen:

- Looping through and referencing a large amount of cells, some of which are empty, while looking for something.
- A search might go past the end of a column or row if the extent of the data is not known.
- Creating large areas which extend past where the data is located.

```
/* INCORRECT CODE:

Area testArea = ws.CreateArea(1,1,75,75);

// If the entire area is not populated with data, then this results in empty
cells being created,
// as they are a part of the area. */
```

Why this method causes issues:

- Forces ExcelWriter to create cell (and associated) object(s) for each cell referenced. For large quantities of unnecessary cells, this can quickly result in performance issues.

Solutions:

- Don't vaguely guess how much area a data set might take, try to narrow down the area as much as possible towards only populated data.
- Start with the range of populated cells so you are only looping through theirs.

```
// POSSIBLE CORRECT CODE:

Area testArea = ws.CreateArea(1,1,20,4);

// Create areas that are as accurately sized to the data as possible, as opposed
to vastly overscaled.
```

2.) Empty cells in the input file

What might cause this to happen:

- Sometimes people create template files using previously populated reports by clearing out values.
- Templates with large numbers of blank cells can be identified by either the scrollbar, which may be suspiciously small for having no cells with data, or by the size of the file, which may be larger than a normal empty template.

Why this method causes issues:

- Just clearing out the values and formatting does not mean the cell has been removed. ExcelTemplate does not overwrite existing cells while passing values into the template file, it just pushes them down.
- Template files that previously had lots of values that are not properly cleaned out quickly become bloated.

Solutions:

- Identify blank cells. This can be done by hitting ctrl+f, then pressing the button 'Find All'. This will return a list of all cells which currently exist in the file. (If there is a large number of cells, this may take a while.)
- Delete any rows/columns which contain empty cells. Note: The scrollbars may not change after deleting the rows, the best way to check is by saving and re-opening the file.
- Don't apply formatting to ranges of blank cells; instead apply it to either just cells with data markers or to column and row headers.
- If all else fails, start over and create a new, blank, template file.

Apply Styles to Row/Columns, Not Cells/Areas

What might cause this to happen:

- Loop through a large number of cells, setting styles on each cell individually.
- Set group of cells to an area, then apply that style to the area.

```
/* INCORRECT CODE:

Style randDataStyle = wb.CreateStyle();
randDataStyle.BackgroundColor = wb.Palette.GetClosestColor(162, 221, 139);
randDataStyle.HorizontalAlignment = Style.HAlign.Center;
ColumnProperties columnProperties;

System.Random rand = new System.Random();
for (int iRow = 1; iRow < 50; iRow++)
{
    for (int iCol = 0; iCol < 3; iCol++)
    {
        ws.Cells[iRow, iCol].Value = rand.Next(100);

        ws.Cells[iRow, iCol].ApplyStyle(randDataStyle);
    }
} */
```

Why this method causes issues:

- Styles set to areas still apply to each individual cell.
- When styles are set on individual cells, the a copy of the style information is kept for every one of the cells.
- When styles are set on rows/columns, only one copy of the information is kept for each row/column.

Solutions:

- Apply styles to large numbers of cells by setting the style to either rows/columns or to singular data marker cells.
- If more than one style is needed in a row/column, and it can't be handled by switching the style for the other (ie, setting style on a row to change part of a column), try to keep the style that takes a majority of the row/column as the one set by that row/column.

```
// POSSIBLE CORRECT CODE:

Style randDataStyle = wb.CreateStyle();
randDataStyle.BackgroundColor = wb.Palette.GetClosestColor(162, 221, 139);
randDataStyle.HorizontalAlignment = Style.HAlign.Center;
ColumnProperties columnProperties;

// Loop through the 2-4 columns to apply formatting
for (int i = 1; i < 4; i++)
{
    columnProperties = ws.GetColumnProperties(i);
    columnProperties.Width = 100;
    columnProperties.ApplyStyle(randDataStyle);
}
```

When Possible, Use DataReaders Instead of DataTables

What might cause this to happen:

- Although they are less efficient for simpler tasks, DataTables can do everything that DataReaders can, so someone may not realize and may just use a DataTable when a DataReader could have done everything they needed more efficiently.

Why this method causes issues:

- DataTables create an in-memory copy of all the data. They are useful for manipulating data in memory or updating the database, but tends to be rather inefficient for just pulling data out of a database.
- May have a negative impact on Time based performance.

Solutions:

- DataReader is more efficient at pulling out data and displaying it, as it just reads one row at a time out of the database and doesn't keep anything in memory.
- DataTables still have uses beyond those provided by DataReader, so sometimes they are still needed to filter, sort or otherwise manipulate the data before populating the workbook.

Time Related Performance Issues

View the information on performance issues that effect the time a report takes to process. 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.