**FlowDocument pagination with repeating page headers**

By [**Bram Fokke**](http://www.codeproject.com/script/Membership/View.aspx?mid=5787472) | 17 Dec 2008

[C#](http://www.codeproject.com/search.aspx?aidlst=81)[Windows](http://www.codeproject.com/search.aspx?aidlst=94)[Dev](http://www.codeproject.com/search.aspx?aidlst=118)[WPF](http://www.codeproject.com/search.aspx?aidlst=140)[Design](http://www.codeproject.com/search.aspx?aidlst=148)[Intermediate](http://www.codeproject.com/search.aspx?aidlst=153)

A DocumentPaginator that supports page headers, footers, and repeating page headers.

Part of [**The WPF / Silverlight Zone**](http://www.codeproject.com/Zones/WPF-Silverlight/) sponsored by

**Sponsored Links**

窗体顶端

* [Download source code - 3.11 KB](http://www.codeproject.com/KB/WPF/PimpedDocumentPaginator/PimpedPaginator.zip)

**Introduction**

Flow documents are designed to optimize viewing and readability. Rather than being set to one predefined layout, flow documents dynamically adjust and reflow their content based on run-time variables such as window size, device resolution, and optional user preferences. In addition, flow documents offer advanced document features, such as pagination and columns.

Unfortunately, three key pieces of functionality are missing from the FlowDocument pagination functionality:

* Page headers
* Page footers
* Repeating table headers

This article describes how you can provide this functionality using a custom DocumentPaginator.

**Background**

Document pagination is provided by the [DocumentPaginator](http://msdn.microsoft.com/en-us/library/system.windows.documents.documentpaginator.aspx) class. This class can be subclassed to override to provide custom behaviour. But, document pagination is a pain in the behind. I didn't want to write all the behaviour from scratch.

The default Document Paginator used by a FlowDocument is not defined in the System namespace, and can't be subclassed. But, we can create a DocumentPaginator subclass which takes aDocumentPaginator as input and uses it for basic pagination. This approach has been described by Feng Yuan in his [blog](http://blogs.msdn.com/fyuan/archive/2007/03/10/convert-xaml-flow-document-to-xps-with-style-multiple-page-page-size-header-margin.aspx).

**Headers and footers**

The sizes for page headers and footers usually are fixed. Therefore, adding them can be accomplished by subtracting the space needed for headers and footers from the available content area and translating the content area so it does not overlap with the header area. The following source code does exactly this:

 Collapse | [Copy Code](http://www.codeproject.com/KB/WPF/PimpedDocumentPaginator.aspx##)



public class PimpedPaginator : DocumentPaginator {

public override DocumentPage GetPage(int pageNumber) {

*// Use default paginator to handle pagination*

Visual originalPage = paginator.GetPage(pageNumber).Visual;

ContainerVisual visual = new ContainerVisual();

ContainerVisual pageVisual = new ContainerVisual() {

Transform = new TranslateTransform(

definition.ContentOrigin.X,

definition.ContentOrigin.Y

)

};

pageVisual.Children.Add(originalPage);

visual.Children.Add(pageVisual);

*// Create headers and footers*

if(definition.Header != null) {

visual.Children.Add(CreateHeaderFooterVisual(definition.Header,

definition.HeaderRect, pageNumber));

}

if(definition.Footer != null) {

visual.Children.Add(CreateHeaderFooterVisual(definition.Footer,

definition.FooterRect, pageNumber));

}

*// Check for repeating table headers*

*// (...will be described later in this article)*

return new DocumentPage(

visual,

definition.PageSize,

new Rect(new Point(), definition.PageSize),

new Rect(definition.ContentOrigin, definition.ContentSize)

);

}

}

Et voila, headers and footers can be added. This was the easy part.

**Repeating table headers**

Automatically adding repeating table headers is a bit more complicated. Basically, there are four separate problems that need to be solved:

* Finding the tables in the document
* Determining if a table spans multiple pages
* Finding the table header
* Inserting the table header if needed

The first three problems can be addressed in a similar manner. The structure of WPF elements is contained in two different trees: the Logical tree and the Visual tree. The logical tree contains the basic structure of a WPF element, and closely resembles the XAML used to describe the element. This did seem like the most logical place to look. Alas, the logical tree of the DocumentPage produced by the default paginator turned out to be of little use, so I decided to inspect the Visual tree. It's huge, but revealing:

 Collapse | [Copy Code](http://www.codeproject.com/KB/WPF/PimpedDocumentPaginator.aspx##)



-- BEGIN PAGE 0 -------

PageVisual (0)

ContainerVisual (1)

ContainerVisual (2)

SectionVisual (3)

ContainerVisual (4)

ParagraphVisual (5)

ParagraphVisual (6)

LineVisual (7)

ParagraphVisual (5)

ParagraphVisual (6)

LineVisual (7)

ParagraphVisual (5)

ParagraphVisual (6)

LineVisual (7)

ParagraphVisual (5)

ParagraphVisual (6)

ParagraphVisual (7)

ParagraphVisual (8)

LineVisual (9)

ParagraphVisual (6)

ParagraphVisual (7)

ParagraphVisual (8)

LineVisual (9)

ParagraphVisual (5)

ParagraphVisual (6)

LineVisual (7)

ParagraphVisual (5)

RowVisual (6)

ParagraphVisual (7)

ContainerVisual (8)

SectionVisual (9)

ContainerVisual (10)

ParagraphVisual (11)

ParagraphVisual (12)

LineVisual (13)

ContainerVisual (8)

ParagraphVisual (7)

ContainerVisual (8)

SectionVisual (9)

ContainerVisual (10)

ParagraphVisual (11)

ParagraphVisual (12)

LineVisual (13)

ContainerVisual (8)

ParagraphVisual (7)

ContainerVisual (8)

SectionVisual (9)

ContainerVisual (10)

ParagraphVisual (11)

ParagraphVisual (12)

LineVisual (13)

ContainerVisual (8)

ParagraphVisual (7)

ContainerVisual (8)

SectionVisual (9)

ContainerVisual (10)

ParagraphVisual (11)

ParagraphVisual (12)

LineVisual (13)

ContainerVisual (8)

RowVisual (6)

(A lot of entries have been omitted for brevity...)

RowVisual (6)

ParagraphVisual (7)

ContainerVisual (8)

SectionVisual (9)

ContainerVisual (10)

ParagraphVisual (11)

ParagraphVisual (12)

LineVisual (13)

ContainerVisual (8)

ParagraphVisual (7)

ContainerVisual (8)

SectionVisual (9)

ContainerVisual (10)

ParagraphVisual (11)

ParagraphVisual (12)

LineVisual (13)

ContainerVisual (8)

ParagraphVisual (7)

ContainerVisual (8)

SectionVisual (9)

ContainerVisual (10)

ParagraphVisual (11)

ParagraphVisual (12)

LineVisual (13)

ContainerVisual (8)

ParagraphVisual (7)

ContainerVisual (8)

SectionVisual (9)

ContainerVisual (10)

ParagraphVisual (11)

ParagraphVisual (12)

LineVisual (13)

ContainerVisual (8)

ContainerVisual (2)

-- END PAGE 0 -------

-- BEGIN PAGE 1 -------

PageVisual (0)

ContainerVisual (1)

ContainerVisual (2)

SectionVisual (3)

ContainerVisual (4)

ParagraphVisual (5)

RowVisual (6)

ParagraphVisual (7)

ContainerVisual (8)

SectionVisual (9)

ContainerVisual (10)

ParagraphVisual (11)

ParagraphVisual (12)

LineVisual (13)

ContainerVisual (8)

ParagraphVisual (7)

ContainerVisual (8)

SectionVisual (9)

ContainerVisual (10)

(...and so on)

The RowVisual elements are the prime suspects to be a row in a table. Every page with a table contains a bunch of them contained in a ContainerVisual. Since the number of children matches the number of columns in this particular document, this is probably the element we are looking for.

By walking the visual tree, we can now find the answers to our questions. If the last element in a page is aRowVisual, there is a good chance that this table will continue on the next page, so we need to save the header of that table for future use. We can find this header by looking for the first RowVisual in the containing ContainerVisual. Conversely, if a page starts with a RowVisual, this probably is the continuation of a table on the previous page, so we should repeat the table header stored earlier. These methods search for the table rows:

 Collapse | [Copy Code](http://www.codeproject.com/KB/WPF/PimpedDocumentPaginator.aspx##)



public class PimpedPaginator : DocumentPaginator {

/// *<summary>*

/// *Checks if the page ends with a table.*

/// *</summary>*

/// *<remarks>*

/// *There is no such thing as a 'TableVisual'. There is a RowVisual, which*

/// *is contained in a ParagraphVisual if it's part of a table. For our*

/// *purposes, we'll consider this the table Visual*

///

/// *You'd think that if the last element on the page was a table row,*

/// *this would also be the last element in the visual tree, but this is not true*

/// *The page ends with a ContainerVisual which is aparrently empty.*

/// *Therefore, this method will only check the last child of an element*

/// *unless this is a ContainerVisual*

/// *</remarks>*

/// *<param name="originalPage"></param>*

/// *<returns></returns>*

private bool PageEndsWithTable(DependencyObject element,

out ContainerVisual tableVisual, out ContainerVisual headerVisual) {

tableVisual = null;

headerVisual = null;

if(element.GetType().Name == "RowVisual") {

tableVisual = (ContainerVisual)VisualTreeHelper.GetParent(element);

headerVisual = (ContainerVisual)VisualTreeHelper.GetChild(tableVisual, 0);

return true;

}

int children = VisualTreeHelper.GetChildrenCount(element);

if(element.GetType() == typeof(ContainerVisual)) {

for(int c = children - 1; c >= 0; c--) {

DependencyObject child = VisualTreeHelper.GetChild(element, c);

if(PageEndsWithTable(child, out tableVisual, out headerVisual)) {

return true;

}

}

} else if(children > 0) {

DependencyObject child = VisualTreeHelper.GetChild(element, children - 1);

if(PageEndsWithTable(child, out tableVisual, out headerVisual)) {

return true;

}

}

return false;

}

/// *<summary>*

/// *Checks if the page starts with a table which presumably has wrapped*

/// *from the previous page.*

/// *</summary>*

/// *<param name="element"></param>*

/// *<param name="tableVisual"></param>*

/// *<param name="headerVisual"></param>*

/// *<returns></returns>*

private bool PageStartsWithTable(DependencyObject element,

out ContainerVisual tableVisual) {

tableVisual = null;

if(element.GetType().Name == "RowVisual") {

tableVisual = (ContainerVisual)VisualTreeHelper.GetParent(element);

return true;

}

if(VisualTreeHelper.GetChildrenCount(element)> 0) {

DependencyObject child = VisualTreeHelper.GetChild(element, 0);

if(PageStartsWithTable(child, out tableVisual)) {

return true;

}

}

return false;

}

}

If both cases are true (i.e., the page starts with a table row and ends with a table row) and theContainerVisuals of both tables are the same, the table spans the entire page. In this case, we do not need to save the first row as the table header.

This approach works pretty well, but has a drawback: it is not possible to detect the case where one table ends on a page and the next page starts with a new table. Fortunately, this situation is pretty rare, and can be completely prevented by adding a paragraph as a table title before the table (naming your tables is a good idea anyway).

This leaves us with one problem to solve: inserting the table header in the generated page. This is going to take some space. The best solution would be to bump the content on the bottom of the page to the next page. After all, that's what a FlowDocument was designed for. Unfortunately, the contents of a page are generated by the default FlowDocument paginator. Bumping content to the next page would involve rewriting the entire DocumentPaginator, which I wanted to avoid altogether.

We'll have to make some room in the existing page. Since the table header usually is much smaller than the rest of the content (typically about 50 times as small), the easiest solution is to vertically scale down the page content a bit to create some headroom and add the table header to the top of the page. If your table headers aren't huge, the resulting distortion is not noticeable. This is the code to insert the table header:

 Collapse | [Copy Code](http://www.codeproject.com/KB/WPF/PimpedDocumentPaginator.aspx##)



*// Check for repeating table headers*

if(definition.RepeatTableHeaders) {

*// Find table header*

ContainerVisual table;

if(PageStartsWithTable(originalPage, out table) && currentHeader != null) {

*// The page starts with a table and a table header was*

*// found on the previous page. Presumably this table*

*// was started on the previous page, so we'll repeat the*

*// table header.*

Rect headerBounds = VisualTreeHelper.GetDescendantBounds(currentHeader);

Vector offset = VisualTreeHelper.GetOffset(currentHeader);

ContainerVisual tableHeaderVisual = new ContainerVisual();

*// Translate the header to be at the top of the page*

*// instead of its previous position*

tableHeaderVisual.Transform = new TranslateTransform(

definition.ContentOrigin.X,

definition.ContentOrigin.Y - headerBounds.Top

);

*// Since we've placed the repeated table header on top of the*

*// content area, we'll need to scale down the rest of the content*

*// to accomodate this. Since the table header is relatively small,*

*// this probably is barely noticeable.*

double yScale = (definition.ContentSize.Height - headerBounds.Height) /

definition.ContentSize.Height;

TransformGroup group = new TransformGroup();

group.Children.Add(new ScaleTransform(1.0, yScale));

group.Children.Add(new TranslateTransform(

definition.ContentOrigin.X,

definition.ContentOrigin.Y + headerBounds.Height

));

pageVisual.Transform = group;

ContainerVisual cp = VisualTreeHelper.GetParent(currentHeader) as ContainerVisual;

if(cp != null) {

cp.Children.Remove(currentHeader);

}

tableHeaderVisual.Children.Add(currentHeader);

visual.Children.Add(tableHeaderVisual);

}

*// Check if there is a table on the bottom of the page.*

*// If it's there, its header should be repeated*

ContainerVisual newTable, newHeader;

if(PageEndsWithTable(originalPage, out newTable, out newHeader)) {

if(newTable == table) {

*// Still the same table so don't change the repeating header*

} else {

*// We've found a new table. Repeat the header on the next page*

currentHeader = newHeader;

}

} else {

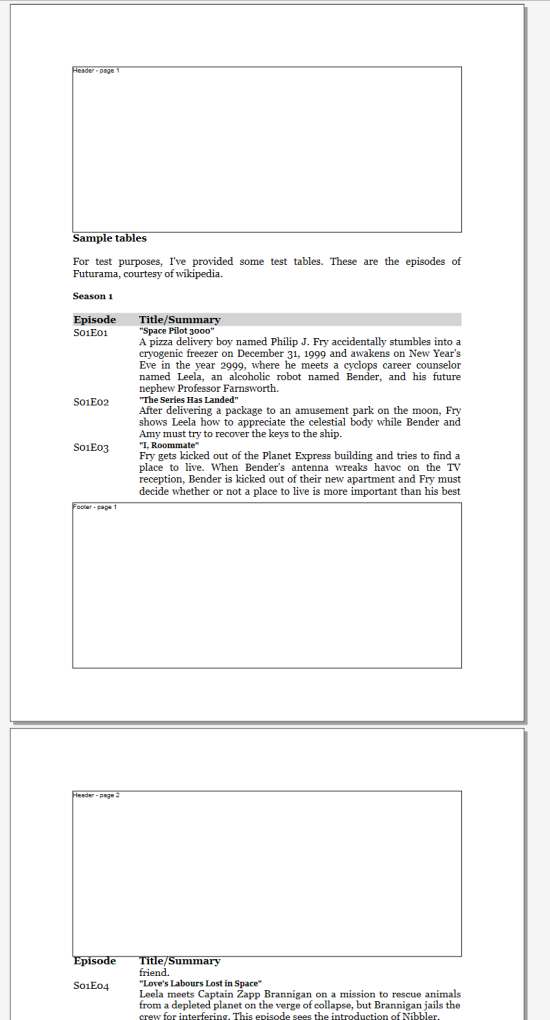
*// There was no table at the end of the page*

currentHeader = null;

}

}

The solution works pretty well, except for one glitch. See if you can spot it:



**Conclusion**

The entire class is included with this article, I hope you can find use for it. Currently, there is one known issue with the implementation: the header background colors are not properly printed when repeated on a new page.

**History**

* 17 December 2008: First version.

**License**

This article, along with any associated source code and files, is licensed under [The Code Project Open License (CPOL)](http://www.codeproject.com/info/cpol10.aspx)

**About the Author**

窗体底端