Exporting Excel files to HTML

Documentation: May, 2010

Web: http://www.tmssoftware.com
Email : info@tmssoftware.com

Table of contents

Introduction

One of the things FlexCel allows you to do is to export your xls files to HTML. Note that as always, we do our best to create the HTML files as closely as possible to the original xls file, but there are some restrictions in HTML that make it not as good as a PDF export. HTML is a "Flow" format, opposed to PDF (which is a "Fixed Page" format), and this means the browser can resize and reposition elements depending on the output medium. With this limitation in mind, we will study now how this is done.

Creating HTML files

Introduction

There are two ways to create an HTML file from inside FlexCel:

1. Using the **FlexCelHtmlExport** component.

This is the most flexible way to create an HTML file, and the lower level too.

2. Using the **FlexCelAspViewer** Component.

This is a component that allows for "drag and drop" display on a web page of an xls file, using ASP.NET 2.0 or newer. Internally it uses FlexCelHtmlExport, and provides some extra functionality to avoid repetitive tasks.

Creating a file using FlexCelHtmlExport

FlexCelHtmlExport is a straightforward component. You would normally set its **Workbook** property to the xls file you want to export, and then call **Export, ExportAllVisibleSheetsAsTabs** or **ExportAllVisibleSheetsAsOneHtmlFile** depending on the type of export you want to do. As always, we will not cover those methods or the properties on FlexCelHtmlExport in detail, since they are described in the reference and it makes no sense to repeat the information here. This document is about the conceptual part of creating HTML files.

Naming the created files
The first thing we need to note is that creating an HTML file is different from creating a PDF or other types of files in the sense that you actually create many files for a single xls file.

Exporting "workbook.xls" to PDF will return in just one file: "workbook.pdf", but it might result in 3 files when exporting to HTML: "workbook.htm", "workbook_image1.png" and "workbook.css"

So we need a way to name the different generated files. By default FlexCelHtmlExport will take the name of the main HTML file as a parameter to the export, and generate the name of the images with the following pattern:

```
<ImagesFolder>\<htmlfilenamewithoutextension>_image<imagenumber>.<imag
eext>
```

This pattern might change, but all in all it gives a good default for images being created. You might use a GUID too as name for the generated images, by changing the **ImageNaming** property.

Now, you might want or need more control than this over the filenames created, and FlexCelHtmlExport gives you that with the **GetImageInformation** event, where you can specify exactly which filename you want for each image or even a stream for each one if you are saving for example the file to a database and not to a file system.

Remember that OnGetImageInformation will be called for every image in the xls file, even if they are not real images. For example, a chart will be rendered as an image, and so it will throw an OnGetImageInformation event. Also, if you have vertical text and the VerticalTextAsImages property is true, then this event will also be called when creating the images with the vertical text.

CSS files are other place where you might create extra files. The HTML standard allows embedding the CSS files inside the generated HTML or to use an external stylesheet, and so does FlexCelHtmlExport. If you decide to go for an external stylesheet, you need to tell FlexCelHtmlExport where to place it, by providing a **TCssInformation** class or a CSS filename to the export methods, that it can use to figure out where to save it.

Embeding the generated HTML inside your own pages
Other thing you that is different in generating HTML from other file formats, is that you might want to get only a part of the html file and not all of it. For example, imagine you want to embed an xls file inside your existing company page. You will not want the whole html file (since embedding <html> tags inside other <html> tags is not valid HTML), but only the part between the <body> tags. And you will want to have the <head> part of the file in a separated place, so you can merge it with the <head> tags in your site.

With FlexCelHtmlExport, you can use the **PartialExportAdd** method and the **TPartialExportState** class for this.

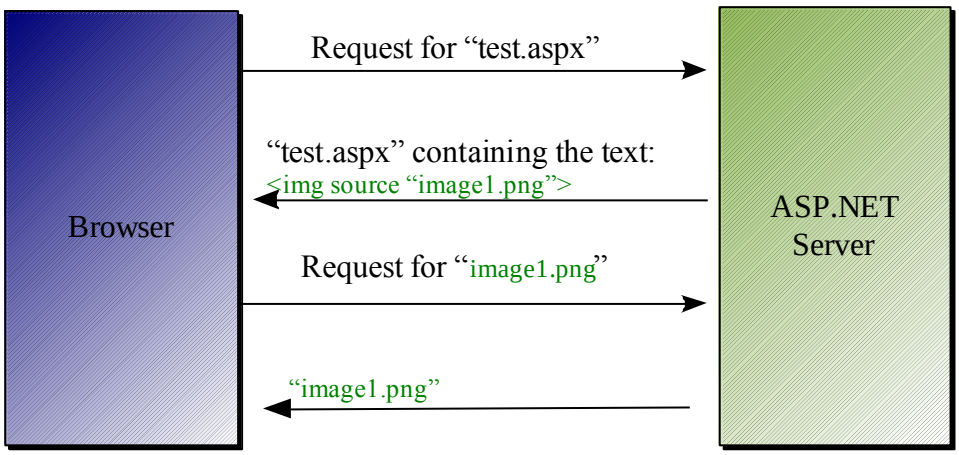
You start by creating a new TPartialExportState instance, where FlexCel will store all the information it needs to create your files.
Then, you call FlexCelHtmlExport.PartialExportAdd() for each sheet or file you want to add to the HTML file, using the same TPartialExportState instance as parameter.
Once you have added all the sheets and files you want, you can use the methods in TPartialExportState to get the individual parts of the HTML file.

Creating a file using FlexCelAspViewer component

The FlexCelAspViewer component is a component for .NET 2.0 or newer, that allows an easy integration of the xls file with your ASP.NET site. Internally it uses FlexCelHtmlExport, and as explained in the section above, it uses the **PartialExportAdd** method and **TPartialExportState** class to inject the different parts of the HTML file inside the ASP.NET file you are designing.

While normally really straightforward to use (just drop it and set its properties), there is one thing to take in account, and this is images.

FlexCelAspViewer needs a way to feed the images back to the browser. While it can send the HTML text back to the main stream when the browser requests an ASP.NET page, it has no way to send the images to the browser, when the browser requests them after reading the HTML file. Remember the browser will make many requests, the first for an aspx file, and then for the individual png/jpg/gif files, as shown below:



In this example we need a way to provide "image1.png" from the ASP.NET server. On the request for the page (test.aspx) we can create the report and send back the HTML, but we need to persist all images here, so when the browser asks for them we know how to get them.

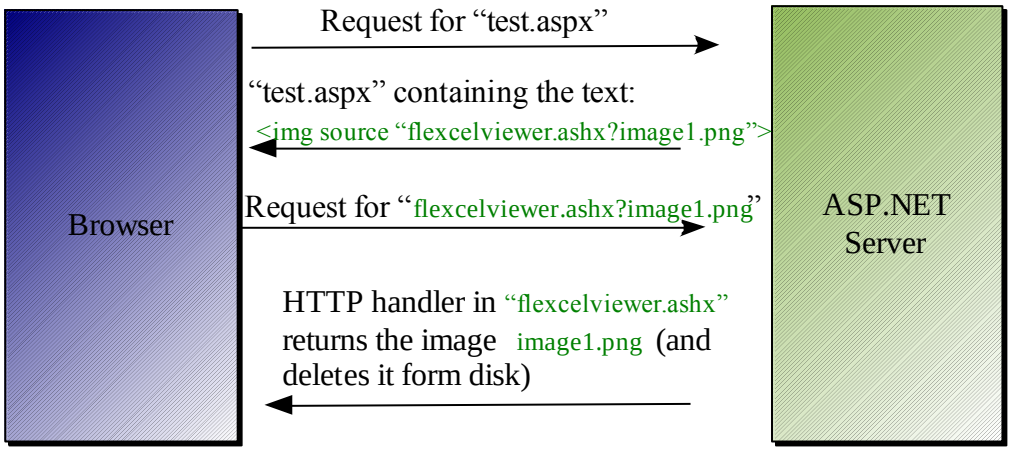
We provide three operating modes for images so they can be sent to the browser, and they are controlled by the **ImageExportMode** property:

TImageExportMode.TemporaryFiles
This is the simplest mode, and when using it, FlexCel will output all the images to a temporary folder when the browser asks for the page. The image links in the main HTML file will link to this folder, and so the browser will be able to get them when it needs them.

Images will be named with a GUID, so even if two different browsers ask for the same report at the same time, they will get different images.

When working in this mode you will need to implement a "Garbage collection" of images older than a given timespan, in order to avoid infinite grow of temporary images. You can do this with some scripts on the server, or use the ImageTimeout property in FlexCelAspViewer.

TImageExportMode.UniqueTemporaryFiles
This mode is not actually suited for wide use, but can be used in controlled environments, where you can test that the browsers work fine. Not all the browsers will work in this mode. Older browsers might ask for the image twice, and will not get it the second time
On this mode, as in the first one, images will be saved to a temporary folder when the main page is requested by the browser. But, instead of links to the image, the generated HTML will contain links to HTTPHandlers, as shown below:



The advantage over the first method is that images will be **deleted by the HTTP handler once they are served.** This provides a more scalable solution, since you do not rely so heavily in the garbage collection, but on the other hand if a browser re-asks for the same image it will not find it.

Important note: It is very important to realize that while this method minimizes the need for garbage collection on temporary images, it does not eliminate it. You will still need to periodically delete the images from the images folder!

The reason why there could still be images left even when they are deleted by the HTTP handler is easy, and it is that the HTTP handler might never be called. Let's imagine the following "broken flow":

As in the picture above, the browser requests an html file from the ASP.NET server. And as above, the ASP.NET server returns the html file with links to the handlers for the images.

But before the browser actually requests the handlers in the HTML file, the user loads other web page. In this case, the "flexcelviewer.ashx" will never be called, and the image never deleted.

Now let's imagine a malicious user, trying a "denial of service" attack to your server. He could make a script that asks for a page in your server, but never calls the handlers, creating lots of temporary files that will never be deleted. In fact, just refreshing his browser manually hundreds of times will get this effect. In each refresh a new html page is generated (along with all the corresponding temporary images), but as he presses refresh again before the page is loaded, the image handlers will not be called and the images will not be deleted.

So, while this method provides a better mode for normal use (where the user requests the page and waits for the images), it is no better to prevent malicious attacks. FlexCelAspViewer provides a property, MaxTemporaryImages, that can be used to mitigate this effect, by deleting older files once you have too much temporary files in the disk. It is recommended that you set MaxTemporaryImages to a reasonable value.

TImageExportMode.CustomStorage
This is a more advanced method that allows you complete control over how the images will be delivered to the browser. It works by creating links to ASP.NET HttpHandlers and allowing you to customize those HttpHandlers to your needs.

It is in some ways similar to the UniqueTemporaryFiles method in that the generated HTML file will have links to HTTP Handlers instead of images, but it is your job now to implement it.

There are two things you need to implement here:

1. You need to specify the image links for the HTML file, providing the parameters you need in order to get the images when the http handler is called.

2. You need to implement the HTTP handler that will return the images to the browser. It could use temporary files as the second mode, or it could cache them in memory, a database or wherever you want.

In order to specify the image links you need to assign the event ImageLink in the FlexCelAspViewer component.

You have complete freedom in how to implement your HttpHandler as long as you implement the IhttpHandler interface, but you can derive your class from FlexCelHtmlImageHandler to avoid writing the basic code. Code in FlexCelHtmlImageHandler has been adapted from the article:

http://www.hanselman.com/blog/PermaLink,guid,5c59d662-b250-4eb2-96e4-f274295bd52e.aspx

and it provides the basic boilerplate common to all Http Handlers. We would like to specially thank Scott Hanselman for it.

You might also take a look at the implementation of UniqueTemporaryFilesImageHandler class in the code in order to get an idea of how a real implementation might look like.

When implementing the handler, you will need to use the **SaveImage** event in FlexCelAspViewer in order to get the images you will need to serve later.

Important note: Even when it is mentioned in the FlexCel reference, we think this is important enough to repeat here: Take a lot of care when creating your own custom handler. A naive implementation might provide a malicious user access to any file in the server, if you fail to validate the parameters correctly.

Improving the image handling in TemporaryImages mode
While TemporaryFiles is probably the best mode for general use, it has the drawback that can create too many images in heavy loaded sites. Each time a user requests a file, there will be an image created for each of the images in the xls file, and named with a unique name so it doesn't crash with the other users. This means that if you have 7 users asking for the same document at the same time, and the document has 8 images, this mode will generate 7x8 = 56 different images. If of all those images only one is dynamic (for example a chart), and the other 7 are static (for example the company logo) you can minimize the number of images created by supplying an **GetImageInformation** event, and setting the ImageFile parameter to null (so no image is created) and the ImageLink pointing to a place where you have the static images.

For example, you could have this event handler:

```
private void htm_GetImageInformationOverwrite2(
                object sender, ImageInformationEventArgs e)
{
    string ShapeName = e.ShapeProps.ShapeName;
    if (ShapeName == "logo")
    {
       e.ImageFile = null;
       e.ImageLink = "images/logo.gif";
    }
}
```

And manually place logo.gif in the images folder, so it does not have to be generated each time.

You can do even more things. If you set the ImageFile parameter to an absolute value (for example "c:\www\images\logo.png"), and the link to point there (for example "/images/logo.png") each time you create a file those images will be recreated so you do not have to care about manually updating them.
The only issue here is that when two users ask for the same image at the same time, one will get a sharing violation because the image is being created by the other thread.

FlexCelHtmlExport has a property "IgnoreSharingViolations" that is true by default and fixes this issue. When a file is locked by another thread, FlexCelHtmlExport will assume this is because there was other thread writing the same image, so it will just not write it.
In the example, if two threads try to write to "c:\www\images\logo.png" at the same time, the first one will succeed and the other will silently fail and continue with the rest of the file. When the browsers then ask for "/images/logo.png" both browsers will be server with the same image that was written by the first thread.

Customizing the generated HTML

General Customization

Customization is a very important part of creating HTML files, since differently from xls or pdf files, HTML files normally have to be integrated with an existing site. This means the style of the generated pages must match the general look and feel of the whole website, and we tried not to cut any corners in letting you do so.

Customizing the raw HTML

There is one property, ExtraInfo where you can add extra HTML in different parts of the document. For example, you can use this property to add META tags to your file including the keywords for searching.

Customizing the Sheet Selector

When exporting a workbook to different HTML files we offer three ways in which you can customize the tabs that represent each sheet:

1. You can get basic customization of colors and properties by changing the CssTags property in the TStandardSheetSelector class.

   The CssTags is a collection of macros that will be replaced in the style definition of the Selector, allowing to change the width of the selector (when placed at the right or at the top, the background color, etc). There is detailed information on the available properties you can change in the FlexCel reference.

2. If the basic customization is not enough, you can completely redefine the css properties in a TStandardSheetSelector class. There is a general property for the styles that apply to all the position in the selector (CssGeneral) and then customized properties for all the other positions.

   For example, you could set border black for all CssTabs with CssGeneral style, and then redefine the border as blue when the tab is on the left by using the CssWhenLeft property.

   You can use the built-in macros when defining your CSS properties and you can even define your own. For example, you could define:

```
CssWhenLeft.Main = "float: left; width: <#width>";
```

And then set

```
TStandardSheetSelector.CssTags["width"] ="100px"
```

As explained, you can even define your own macros. For example you could use

```
CssWhenLeft.Main = "float: <#floatpos>;";
```

And then add "floatpos" to the CssTags array. This way you will be later be able to change your own styles easily. The syntax for using a macro in a CSS definition is "<#macroname>", and then you need to add it to the CssTags array.

3. Normally methods 1) and 2) should be enough to handle most needs, but if you need a completely new way to display your tags, you can just define your own SheetSelector by deriving it from the class TSheetSelector. You will need to override the abstract methods in this class to provide the behavior you need. In fact, this is the way TStandardSheetSelector is defined, and you can look at its code when creating your own SheetSelector class.

Customizing the Fonts

Fonts used in the HTML file are the same that the ones used in the xls file, and this might bring some compatibility issues if you expect your file to be shown in a website where it can be opened by people all over the world.

So it is recommended that you stick to standard fonts, like Arial or Times New Roman on the xls files you want to export. But if the files already exist and have non standard fonts, you can use the HtmlFont event to convert the fonts to standard typefaces. You can search for an example on this in the Export HTML demo.

Important note: The quote character FlexCel uses in style tags is the single quote ('). So, if you need to quote your font names because they have spaces, use a double quote (") so the style declaration is not affected.
For example, a style declaration in FlexCel might be: style = 'font-decoration: "my font"; ' You need to use double quotes around "my font" so it does not end the style declaration.

Export to HTML and FlexCel recalculation

Look at the notes in "Using FlexCelPdfExport" on the recalculating issues, this applies to HTML too.

Using Relative Hyperlinks

You can create links in Excel, both inside cells or on images, and they will be exported to html. But there is a small issue in that in Excel you can only enter absolute URLs. For example, you can enter: "http://www.tmssoftware.com/docs/index.htm", but you cannot enter just docs/index.htm. If you want your websites to be "site independent", you need to replace those URLs with relative ones.

You can do this using the "BaseUrl" property. If for example you define"BaseUrl" to be "http://www.tmssoftware.com/", whenever a link starts with that text, the text will be removed. In our example, the link:
"http://www.tmssoftware.com/docs/index.htm" will be converted to
"docs/index.htm"