Creating Special Page Headers

OWrite and JS-OWrite page headers and footers can contain formatted text plus inline pictures, which is sufficient for most types of document. However, more sophisticated text wrapping or inserting of complex objects such as tables are not supported. For some types of document this may be limiting.

The solution is to use header/footer template documents where the content of the document is returned as a high-resolution image to an inline-picture field in the main document's header or footer. The inline-picture field can use the external source feature so the main document's size is not bloated by the high-resolution representation of the custom header/footer.

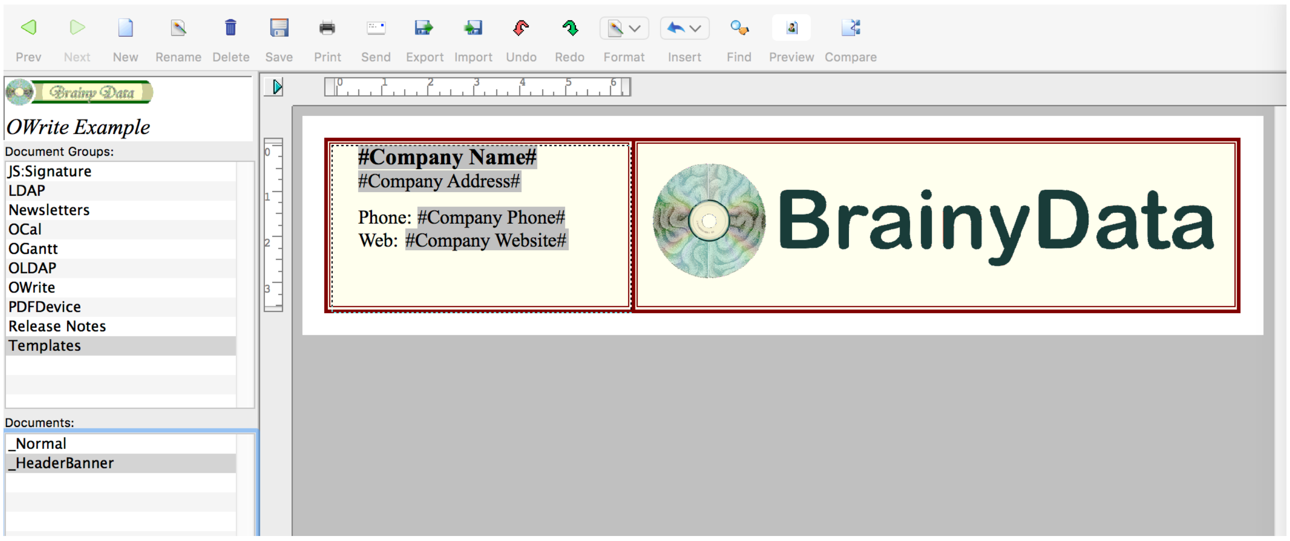
The implementation of this feature requires four distinct actions.

1. Create a custom header/footer template document.
2. Create a method for evaluating the template and producing a high-res image.
3. Provide an interface for inserting an inline-picture with an external source.
4. Implement an event for fetching the image data when OWrite requests it.

We have implemented a custom-page-header for our simple JS-OWrite example (see OWriteDocumentManager.lbs) and we will use this example as the basis for explaining the process.

1. Create the template document

In our example we created a template document using the OWrite document manager. The template contains a single table field with two columns. The left column contains fields for merging company details and the right column is populated with a logo that was pasted from the clipboard. We can use the interface provided by the document manager to insert and edit the required items.



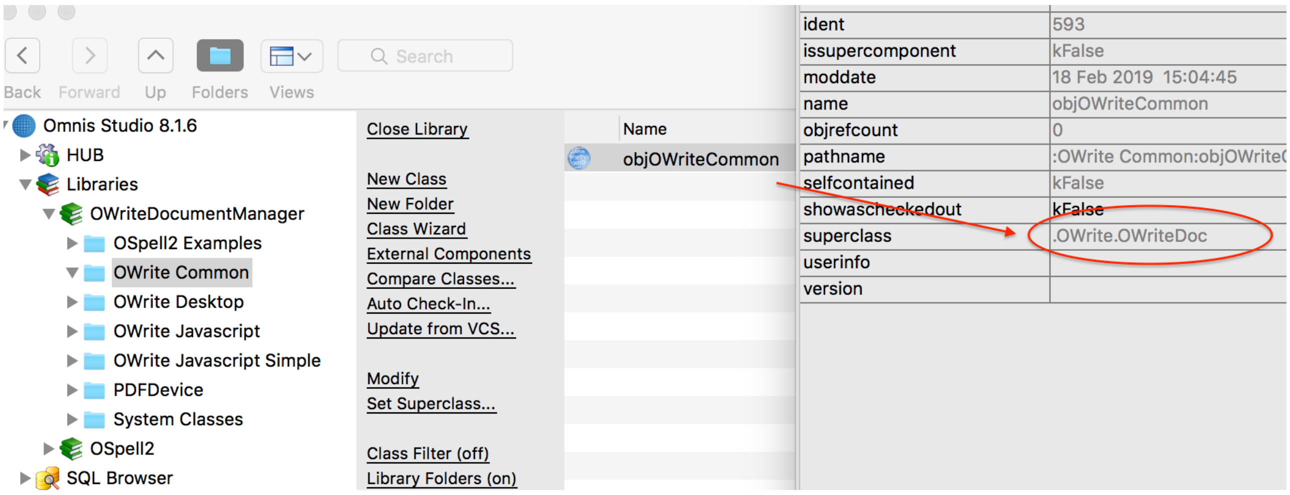
#Company Name#
#Company Address#

Phone: #Company Phone#
Web: #Company Website#

The calculated fields in the examples contain calculations such as *$cinst.$eval('CompanyName')*. When we produce the high-res image from the template in the next stage, we must first evaluate the document within the correct context.

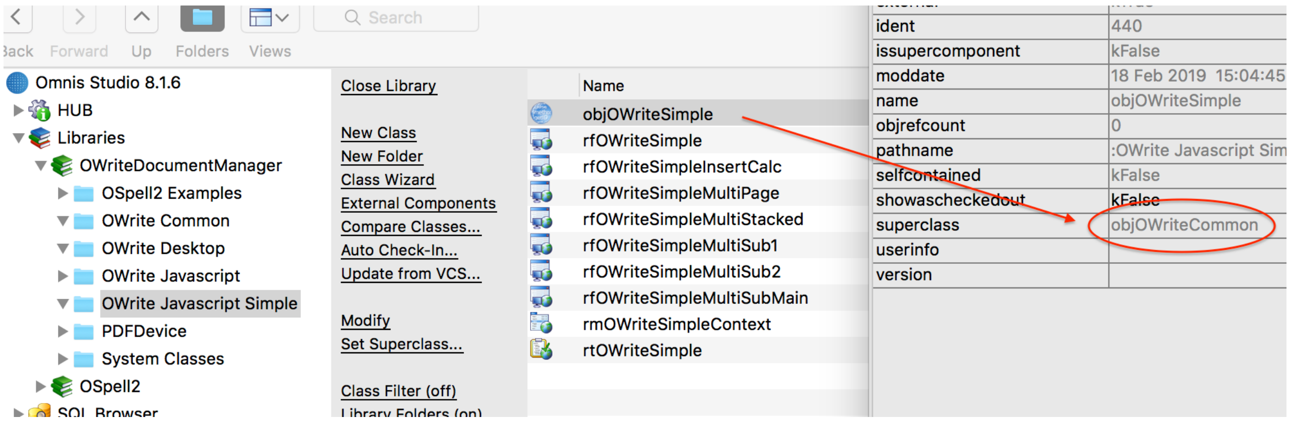
2. Produce the high-res image data

To evaluate and create the high-res image we created a new object class *objOWriteCommon* which can be found inside the folder *OWrite Common*. The object class inherits from the external non-visual OWrite document object which allows it to load and evaluate OWrite documents.



ident	593
issupercomponent	kFalse
moddate	18 Feb 2019 15:04:45
name	objOWriteCommon
objrefcount	0
pathname	:OWrite Common:objOWrite
selfcontained	kFalse
showascheckedout	kFalse
superclass	.OWrite.OWriteDoc
userinfo	
version	

The Simple JS-OWrite example already implemented an object class called *objOWriteSimple* for evaluating documents. In order for this example to have access to this new feature, we have changed the superclass of this object to *objOWriteCommon*. If we wanted to add this feature to the OWrite desktop examples we could do a similar thing.



ident	440
issupercomponent	kFalse
moddate	18 Feb 2019 15:04:45
name	objOWriteSimple
objrefcount	0
pathname	:OWrite Javascript Sim
selfcontained	kFalse
showascheckedout	kFalse
superclass	objOWriteCommon
userinfo	
version	

The objOWriteCommon object implements four methods.

Method $eval()

This method will be called by the calculated field when we evaluate the document template.

Please refer to the example source for further details.

Method $getHeadeBanner()

This method does the actual work of loading, evaluating and generating a high-res image from the document template.

One of the key tricks is to set both the *$screendpi* and *$printdpi* to 288 pixels per inch. This not only increases the resolution and thus quality of the final image that is produced, but 288 is also the *lowest common multiple* of both 96 and 72, thus ensuring a smooth scaling of font sizes and pixels on both windows and macintosh platforms.

Another key trick is the new feature we added to *$picturefrompage()*. This method can now receive zero values for the width and height of the final image, indicating we want a full resolution image (no scaling down), plus an additional parameter that specifies the ID of an OWrite object which limits the image to the specified object (in our case our table field).

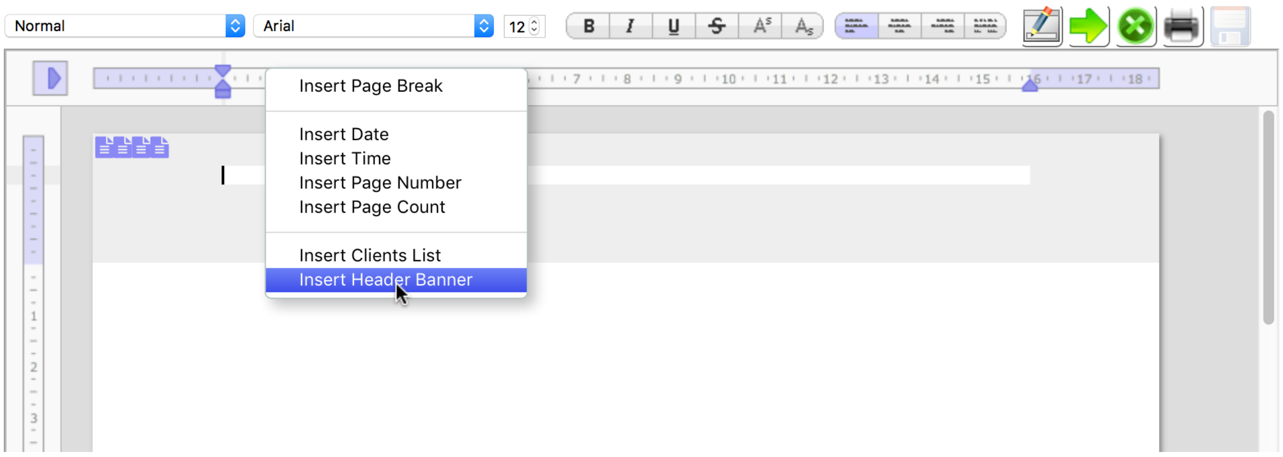
Please refer to the example source for further details.

Method $getHeaderBannerWidth()
Returns the image width in OWrite units (1/1000mm)

Method $getHeaderBannerHeight()
Returns the image height in OWrite units (1/1000mm)

3. Inserting the external source image

In our JS-OWrite simple example we chose to add an option to the context menu which will insert the image.



Inserting the image will execute our $owriteInsertObject client method (see case kWriObjTypePict near the end of the method).

As well as inserting the object, this method also updates the indents, page margin and header margin to make room for the banner in the header area.

```
Case kWriObjTypePict     ;; change20190207_1 added for adding header banner to document
   If pObjSubtype='HEADER_BANNER'
      ;  Insert picture with external source to evaluate and fetch image of the _HeaderBanner template document

      ;  Do $cinst.$objs.OWrite.$insert(kWriObjTypeText,"12345",kWriInsertAfter,con("Insert",kWriObjTypeText)) Returns ok
      ;  calculate image size from sample header banner row     ;; (size must be in 1000/mm)
      Calculate origWidth as ivSampleHeaderBanner.width
      Calculate origHeight as ivSampleHeaderBanner.height

      ;  prepare paragraph for banner     ;; (if we are inside the header)
      Calculate $cinst.$objs.OWrite.$curleftindent as -2     ;; 0.5-$cinst.$objs.OWrite.$leftmargin
      Calculate $cinst.$objs.OWrite.$currightindent as -2     ;; 0.5-$cinst.$objs.OWrite.$rightmargin
      Calculate $cinst.$objs.OWrite.$::topmargin as 5.0
      Calculate $cinst.$objs.OWrite.$headermargin as 0.5
      Do $cinst.$objs.OWrite.$updatedocprops.$assign(kTrue)
      ;  Do $cinst.$objs.OWrite.$setselection("h0.p0.c0","h0.p0.c5")

      ;  prep params for inserting a picture object     ;; Note: we use the external image src feature so the image data is not stored in document data
      Do $cinst.$objs.OWrite.$initparams(kWriObjTypePict) Returns params
      Do $cinst.$objs.OWrite.$addparams(params,"ObjFormat",kWriObjFmtInline,"ObjAlign",kWriObjAlignCenter,"ObjFlags",'LockAspect')
      Do $cinst.$objs.OWrite.$addparams(params,"ObjOrigWidth",origWidth,"ObjOrigHeight",origHeight,"ObjWidth",origWidth,"ObjHeight",origHeight)
      Do $cinst.$objs.OWrite.$addparams(params,"ObjDataSource","$cinst.$getHeaderBanner()","ObjName","HeaderBanner")

   End If
```

When the image with the external source has been inserted, OWrite will generate the evGetDataFromSrc event, which brings us to the next stage.

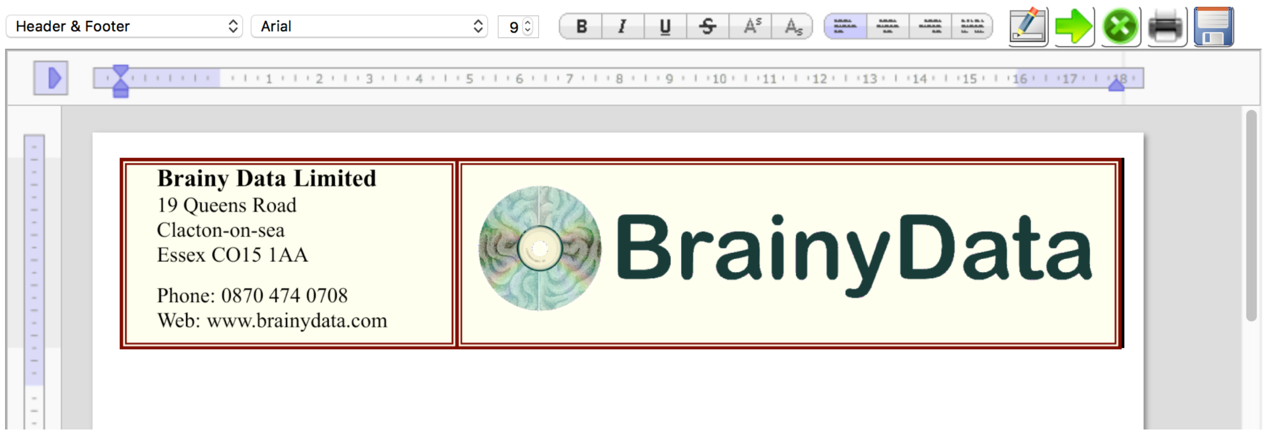
4. Fetch the external source data

In our example we have actually put the cart before the horse in that we built the header banner image when the remote form constructed. We did not wait for OWrite to request the image first. If the image is only required some of the time, waiting for the request to generate the image would probably be more efficient. However, in our case, *rfOWriteSimple.$construct()* calls *$cinst.$serverBuildHeaderBanner()* which in turn calls *objOWriteSimple.$getHeaderBanner()*. Our *$serverBuildHeaderBanner()* method converts the image to PNG and then populates our ivSampleHeaderBanner row variable with the data and details of the image. This row variable can be accessed by our client methods when the evGetDataFromSrc event is generated.

Our event code simply calls

    $cinst.$objs.OWrite.$setdatafromsrc(pObjID,ivSampleHeaderBanner.pngImage)

and we are done.



Brainy Data Limited
19 Queens Road
Clacton-on-sea
Essex CO15 1AA

Phone: 0870 474 0708
Web: www.brainydata.com

BrainyData

In order to return the horse before the cart, we could call *$serverBuildHeaderBanner()* not from *$construct()*, but from our *evGetDataFromSrc* event and than execute *$setdatafromsrc* in our *$serverBuildHeaderBanner_return* client method.

Document History

26 February 2019:	corrections
19 February 2019:	first publication