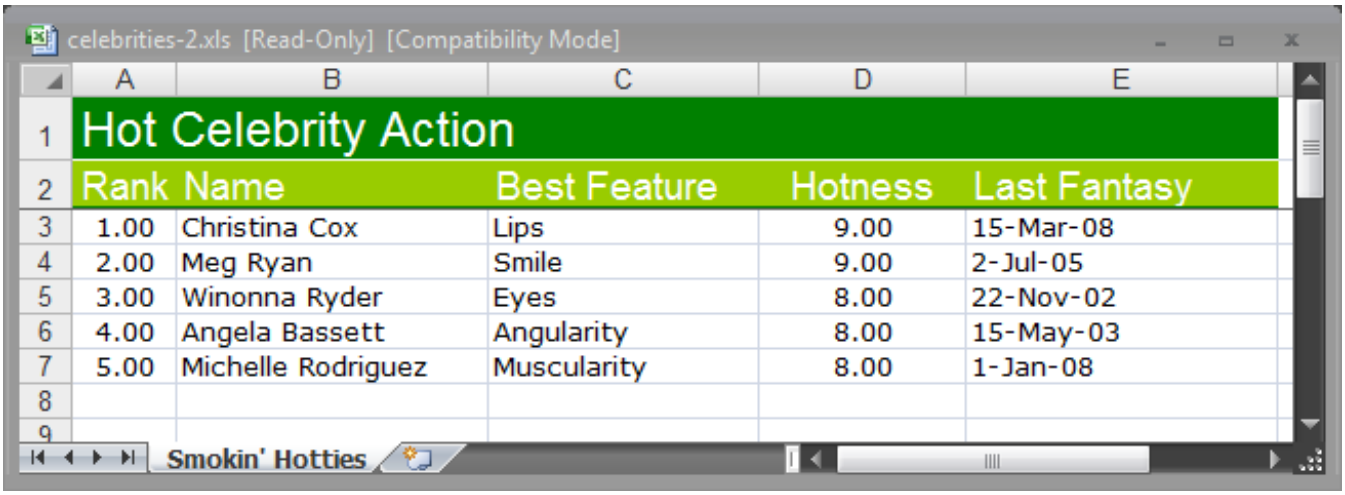
Cell Style Caching Issue With POIUtility

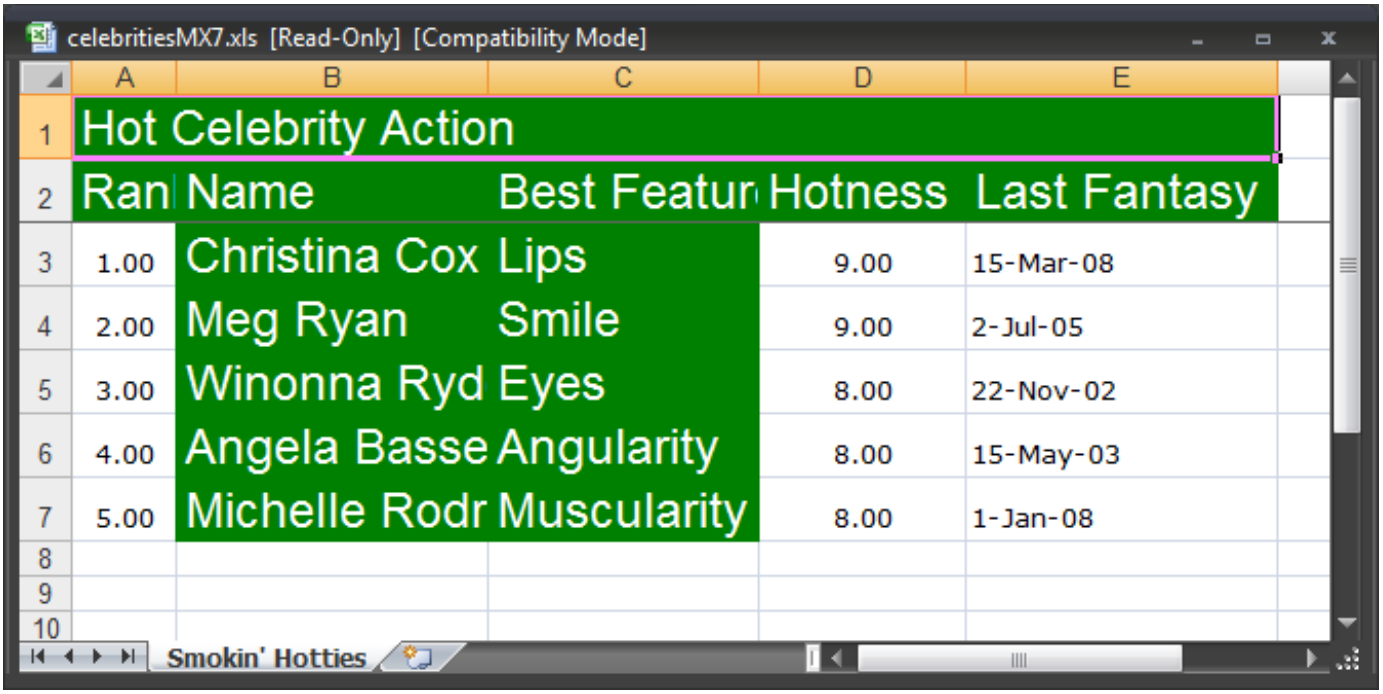
Posted At : September 17, 2008 10:57 PM | Posted By : Nathan Mische
Related Categories: ColdFusion, Java

I've been using Ben Nadel's POIUtilty for a while now. I really like the custom tag library he has built for building Excel spreadsheets, particularly the way he has abstracted the POI cell formatting options out to CSS. Last week however, I ran into a strange issue while running the latest release on CFMX 7. Cell styles were not being properly applied and it seemed to be related to the CSS style caching functionality. What was even more strange was that the code ran fine on CF 8. Below are samples of the correctly styled spreadsheet genrated by CF 8 and the incorrectly formatted spreadsheet generated by CF 7.

CF 8



CFMX 7



So today I finally got a chance to look at the code and was able to quickly spot the issue. The cell styles are held in a structure. This structure is then used by the POIUtitility custom tag code to generate an instance of the HSSFCellStyle class. In order to reuse instances of HSSFCellStyle the code caches instances based on the hashCode of the style structure. Below is brief, simplified, snippet of code to give an idea of what it does.

```
<cfset cache = structNew() />

<cfset style = {font = "Arial", color="blue" } />

<cfset hssfCellStyle = workbook.createCellStyle() />

<cfset hssfCellStyle = CSSRule.applyToCellStyle(style,workbook,hssfCellStyle ) />

<cfset cache[style.hashCode()] = hssfCellStyle />
```

The issue with this approach is that hashCode is not guarenteed to be unique. If you take a look at the spec of javal.lang.Object, the general contract of hashCode states:

> "It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables."

In other words, two different structures may generate the same hashCode. This can explain why CFMX 7 and CF 8 generated different results; the hashing algorithim had obviously been improved from Java 1.4 to Java 1.6, resulting in more unique values. However, given the general contract of hashCode there is no way to guarentee there will not be a similar issue in CF 8 using the current caching implementation.

The fix I came up with was to use a Java Hashtable instead of a Struct to hold the cache. I could then use the style structure instead of its hash as the cache key. Here is what the example above would look like using a hashtable:

```
<cfset cache = createObject( "java", "java.util.Hashtable").init() />

<cfset style = {font = "Arial", color="blue" } />

<cfset hssfCellStyle = workbook.createCellStyle() />

<cfset hssfCellStyle = CSSRule.applyToCellStyle(style,workbook,hssfCellStyle ) />

<cfset cache.put(style, hssfCellStyle) />
```

Reading from the cache becomes a little tricky as you have to deal with nulls. Instead of a simple StructKeyExists call you now have to do something like the following:

```
<cfset cellStyle = cache.get(style) />

<cfif IsDefined("cellStyle")>

    ...

</ciff>
```

This is a decent trade off for a working cache I think.

I was familiar with this from my reading of the Effective Java Programming Language Guide by Joshua Bloch, which talks about implementing hashCode. I highly recommend this book to anyone who does even a little work with Java.