Article
Sean Connelly · Aug 21, 2017 3m read

Handling images with Caché & JSON, and why 57 is a magic number

If you want to dynamically serve images as a property of JSON then there is no perfect encoding solution. One method used frequently is to Base64 encode the image. Whilst there are some negatives to doing this, such as data inflation, there are some positives to working with Base64 images inside the browser.

Let's say you have an image placeholder on a web page...

```
<div id="image-container"><div>
```

And you fetch a JSON message from the Caché server containing the image as one of its properties...

```
var msg = JSON.parse(client.responseText);
```

Without needing to decode the image data you can create an img element and append it directly to the place holder...

```
var img=document.createElement("img");
var src="data:image/jpg;base64," + msg.Image
img.setAttribute("src",src);
document.getElementById("image-container").appendChild(img);
```

and the Image will display.

Here's a jsfiddle example...

http://jsfiddle.net/owqwb9b4/

The parallel question to this is how to safely convert the image to Base64 in the first place. One option is to use the next release of the Cogs JSON library which will be out this week. The new release will add even more ways to work with JSON, as well as supporting stream properties, and auto-converting binary streams to and from Base64.

If doing this from scratch, then converting an image stream to Base64 seems simple enough...

```
$system.Encryption.Base64Encode(obj.ImageStream.Read(12000))
```

except that there are a few problems to consider.

The first thing is that Base64Encode is expecting a string. This is fine if you have long string support enabled, and all of your images are smaller than @ 3/4 of 3.6MB

To get around this limitation, the image has to be encoded in chunks.

Base64 works by taking 3 bytes and converting those 3 bytes into 4 bytes using an algorithm that ensures the 4 bytes are all in a safe ASCII range. Where an input string is not divisible by 3, the algorithm appends one or two zero bytes, which are converted to = or ==, which is why you often see a Base64 string end in these values.

The take from this is that encoding in chunks will produce odd results if each chunk is not exactly divisible by 3 (otherwise you end up with = and == values inside the string). So make sure the read length has a value such as 333.

There is also another issue. By default, the Caché Base64Encode method outputs the encoded string with line breaks every 76 characters. These line breaks will make the JSON invalid, so either you have to remove the line breaks, or escape them, neither of which is pretty and for me always seems to break the Base64 at the other end. Not only will it break the JSON, but the line breaks at the other end will also break the JavaScript img value.

Fortunately, if you are on a newer version of Cache then there is now an argument that will disable the line breaks. Unfortunately, it's not in older versions (such as 2014 that I have tested this on.)

There is however, a simple workaround that will be backwards compatible, which is to use a read length of 57 (or less). This ensures each chunk is under 76 characters in length and avoids the line break being added.

So a manual solution might look something like this...

```
 write """Image"":"""
 do ..Image.Rewind()
 while ..Image.AtEnd=0 { write $system.Encryption.Base64Encode(..Image.Read(57)) }
 write """"
```

#JSON #Object Data Model #Caché

Source URL:https://community.intersystems.com/post/handling-images-cach%C3%A9-json-and-why-57-magic-number