Article
Ben Spead · Aug 25, 2016 4m read

Writing forward compatible JSON in 2016.1

Background

Caché 2016.1 introduced some very powerful JSON features. Unfortunately, the syntax used in the 2016.1 release is going to be changed in the 2016.2 release to bring it closer to Caché ObjectScript syntax. This will mean that code that works in 2016.1 may not be forwards compatible. See this announcement and this article for more details about the changes.

In order to try to minimize the volume of code which would not work in future Caché versions, the following recommendations and tools are being used internally within InterSystems for internal application JSON code being written in 2016.1.

These are being published here as they should be useful to our customers as well. Following these recommendations and adapting the macros linked below should make a future upgrade from 2016.1 to 2016.2 straightforward and not require any time-of-upgrade code changes for JSON code.

Use of ()s

In 2016.1 the requirement of when ()s were and were not required when writing JSON with Caché ObjectScript was inconsistent. Therefore, future versions of Caché will be requiring ()s around a much larger percentage of Caché ObjectScript expressions when they are used as values in a JSON object.

Recommendation: When writing any Caché ObjectScript as part of JSON, always put the Caché ObjectScript in ()s (in other words - if it isn't valid JSON, then put it in ()s). This should include Caché ObjectScript variable names. For example:

```
set a = {"b":($get(c)+$get(d))}
set b = ["b",(first_last)]
```

In 2016.1, literal JSON string values (for the sense of "string" and "value" defined at http://www.json.org/) were escaped using Caché ObjectScript escaping (i.e., " is escaped as "" within a string). This meant that not all valid JSON would be valid as an object initializer. Therefore, escaping of literal string values now follows JSON rules rather than Caché ObjectScript . A JSON value enclosed in () will behave the same way on both versions - the parentheses indicate "the stuff in here is Caché ObjectScript, not JSON" - and therefore should be used for literal values containing control characters, double quotes, and backslashes.

Recommendation: When defining any literal values containing a character that must be escaped in JSON, represent the literal as a Caché ObjectScript string with Caché ObjectScript escaping (double double quotes) enclosed in parentheses. For example:

```
set b = ["b",("a string with ""special characters""")]
```

Use of System Methods

The change which will have the largest impact on JSON code written in 2016.1 is the move to bring the new Systemmethod syntax (e.g. .$methodname()) in line with standard Caché ObjectScript naming conventions. In order to do this, any current .$foobar() method is going to be replaced in future versions of Caché with .%Foobar() methods (note the change from $ to % and the change of the first character to uppercase). As the new methods are not available in 2016.1, the best way to 'future-proof' an application being written in 2016.1 will be to encapsulate the calls to all current system methods within macros which will have the new syntax that will take affect in future versions. A ***sample*** set of macros can be found on Gist: https://gist.github.com/bspead/bf23a28029e8d389ac39a3a4af3342bd

IMPORTANT DISCLAIMER: These macros are being provided as examples and are in no way supported by InterSystems (and they will not be part of the product). Also, after all instances of an application are upgraded to 2016.2 or higher the macros should be removed – they are not intended for permanent inclusion in an application ’ s code base.

Recommendation: Using the examples in this Gist code snippet, create a set of macros in your code-base which encapsulates system methods.

Recommendation: Any application using 2016.1 with system methods (.$foobar()) for example should call the macros included in cosJSON.inc instead of calling the system methods directly. For example:

```
set object = {"property1":123,"property2":456} set size = object.$$$jsonSize()
```

Use of %Object and %Array classes

The naming of the %Object and the %Array classes will be changing in future versions of Caché to %DynamicObject and %DynamicArray. Therefore direct calling to any of the %Object or %Array classes should be avoided. Most uses of these classes are behind the scenes and should not be called directly, but there are a few areas where they may be used. Specifically when creating new dynamic objects or arrays - either empty or from a string. To avoid direct use of the class name you can use the inline expression.

Recommendation: Do not use %New() for creating new objects but instead create them inline. For example:

```
set object = {}
set array = []
```

Recommendation: In order to access a ClassMethod simply instantiate a new dynamic object with an inline expression first. For example:

```
set newArray = [].$$$jsonFromJSON("[1,2,3]")
set newObj = {}.$$$jsonFromJSON({"name":"Ben"})
```

Recommendation: If you use #Dim in order to leverage Studio Assist with your JSON variables you can do so safely with %Object or %Array types and this will not break upon upgrading to 2016.2. However you should be aware that Studio Assist will not recognize those classes in versions greater than 2016.1 and your #Dim statements will need to be updated to %DynamicObject and %DynamicArray in order for Studio to function again for those variables in future versions of Caché

#Caché #Code Snippet #Compatibility #JSON #Tips & Tricks

Source URL:https://community.intersystems.com/post/writing-forward-compatible-json-20161