Article
Daniel Kutac · Apr 13, 2017 3m read

Binding a "regular" Cache object to %DynamicObject and vice versa

Recently, a partner company started to develop an Angular client for their Cache application. Together, we decided to leverage the power of Caché dynamic objects to exchange JSON encoded data between client and server parts. However, we realized that currently there is a gap in Cache JSON implementation that prevents simple use of traditional registered and persistent classes to exposed their data with the same ease as with XML. I wrote a small JSON adapter, that does the job and bridgers the gap. It's purpose is simple expose data described by a regular Cache class in a one-to-one fashion to a %DynamicObject. On the other hand, when a serialized JSON data comes in, it can be easily deserialized into dynamic object and subsequently bound to regular class by the newly created adapter. You can find the adaptor class in the github repository here - https://github.com/dkutac/cache-json-regular-vs-dynamic-object

Here is how it works:

Extend your class with **kutac.JSON.Adapter**.

The adapter class injects two methods to your class. One, called %Bind, and the second, called %Expose.

Let's have a closer look that them now.

%Expose() - use this method when you wish to expose your regular Cache class as dynamic object, so you can then manipulate the content of JSON easily by means of **%DynamicObject** class. For instance, if your class has references to other classes or holds a reference to collections, you can easily filter the data to be serialized by dynamic object API before you call **%ToJSON()** method.

An example of using %Expose method: suppose your regular class is *MyApp.Person extends (%Persistent, kutac.JSON.Adapter)*. You can then instantiate a the class and expose (eventually filter unwanted data out) and serialize.

```
set person=##class(MyApp.Person).%OpenId(123)

set JSON=person.%Expose().%ToJSON()

write JSON
```

simple, isn't it?

%Bind() - use this method when you need to process incoming JSON data. The adapter is written in such way, that it first tries to identify - based on data received - whether there is already an existing instance of the class, if the class to be bound is persistent. If it fails opening existing class instance, it creates a new instance.

The adapter works with default generated IDs as well as with custom ones, based on IdKey index definition when defined. Just make sure that you supply all values needed for unique identification of the class instance. Please note, that for default IDs we use "id" key name. *It is expected that key names reflect exactly names of properties in bound classes!*

suppose you have following JSON string coming:

```
set json={"_id":"2", "Name":"Joe", "DOB":"2001-01-29"}
```

then, you can bind the data to the MyApp.Person class and save it this way:

```
set person=##class(MyApp.Person).%Bind({}.%FromJSON(json),.sc)

$$$ThrowOnError(sc)

$$$THROWONERROR(sc,person.%Save())
```

As you can see, using the adapter is easy, it doesn't by purpose, offer any parameters to customize names of json key/value pairs. As the adapter is a fresh new development, there may be errors, so please feel free to fix them and contribute to the github.

One final note: this adapter has by no means any ambitions to replace and future functionality of Cache, but shall make you life easier for now and make it easy to switch to system provided API when it comes with some future version of Cache.

#Object Data Model #JSON #Caché
