Artículo
Jose-Tomas Salvador · 30 mayo, 2019 Lectura de 2 min

# Serialización y deserialización de objetos en formato JSON (u otro cualquiera)

¿Qué pasaría si pudieras serializar/deserializar objetos en cualquier formato: JSON, XML, CSV,...; siguiendo diferentes criterios, exportar/importar unas propiedades y no otras, transformar valores de una u otra forma antes de exportarlos/importarlos,... y todo ello sin tener que cambiar la definición de la clase? ¿No sería genial si pudieras hacer todo eso?

Bueno, quizás sea un objetivo demasiado ambicioso para cumplirlo al 100% pero, al explorar esta idea, desarrollé muchas clases que pensé que sería bueno compartir. Si deseas probar, cambiar, modificar o mejorar el código, o simplemente echarle un vistazo, puedes hacerlo aquí. También encontrarás una explicación más detallada (consulta Readme.md)
**Debes tener presente que** esto es una prueba de concepto y la realicé durante mis ratos libres, por lo tanto, seguramente no es lo suficientemente robusta o puede mejorarse... pero, solo estaba jugando!....ok, podría haber esperado al lanzamiento del nuevo JSON Adaptor que seguro resuelve muchos escenarios de una manera más limpia, pero... mientras llegaba... :-) ...

Básicamente, lo único que debes hacer es importar las clases. Después eliges una de tus clases persistentes o registradas, o creas una nueva y haces que herede de *OPNLib.Serialize.Adaptor.* Después de eso, sólo debes hacer lo siguiente:

```
set obj = ##class(MyPkg.MyPersistentClass).%OpenId(1)
set json = obj.Export()
do json.%ToJSON()
{"prop1":"value1","prop2":"value"}


set newObj = ##class(MyPkg.MyClass).%New()
do newObj.Import(json)
write newObj.%Save()
write newObj.%Id()  //if it's a persistent object, of course
27
```

Este enfoque todavía no evita por completo la necesidad de modificar la definición de la clase... pero una vez que lo hagas, tendrás 2 métodos que pueden actuar como despachadores para cualquier otra lógica de serialización que quieras inyectar en el futuro... y también, si exploramos la idea detrás de la clase OPNLib.Serialize.Template, será relativamente fácil adaptarla para tener un método de compilación previo a la exportación/importación de datos en cualquier formato de serialización, sin la necesidad de modificar la clase fuente/objetivo.

## *¡Que lo disfrute!*

#Code Snippet #JSON #Mapeo #Modelo de datos de objetos #API REST #Caché #InterSystems IRIS

**URL de**

**fuente:**https://es.community.intersystems.com/post/serializaci%C3%B3n-y-deserializaci%C3%B3n-de-objetos-en-formato-json-u-otro-cualquiera